SCSI initial program loading for zSeries

G. Banzhaf
F. W. Brice
G. R. Frazier
J. P. Kubala
T. B. Mathias
V. Sameske

This paper describes a new kind of initial program loading (IPL) for IBM zSeries® servers. The new IPL protocol expands the set of input/output devices that can be used during IPL to include Small Computer System Interface (SCSI) Fibre Channel protocol (FCP) disk devices (SCSI disks). We begin by describing several new challenges resulting from the use of SCSI disks during IPL, followed by a brief overview of new concepts we have applied to the IPL process to overcome these challenges. We continue with a step-by-step description of the processes executed during SCSI IPL, the tools used, the disk format, the parameters required, and related topics. Since SCSI IPL is supported for virtual machines instantiated by the z/VM® operating system, some unique features of this capability are described. Finally, we describe a variation of SCSI IPL that enables the contents of memory to be dumped onto a SCSI disk.

Introduction

This paper describes a new process for loading and starting an operating system (OS) on IBM zSeries* servers. The loading process is referred to as *booting* or *initial program loading* (IPL).

Traditionally, zSeries input/output (I/O) has been based on devices conforming to the zSeries I/O architecture [1, 2]. Since these I/O devices are controlled by channel programs comprising channel command words (CCWs), they are referred to as *CCW-based* I/O devices. IPL from these devices is called *CCW IPL*. Recently, a zSeries version of the Linux** OS and a Fibre Channel protocol (FCP) channel were introduced, enabling zSeries use of Small Computer System Interface (SCSI) FCP devices [3]. We refer to these as *SCSI devices*. A new IPL process, called *SCSI IPL*, enhances this zSeries SCSI support by allowing the loading and starting of an OS from SCSI disk devices (or SCSI disks). This paper describes SCSI IPL, along with related features such as the ability to load a dump program from a SCSI disk.

Typically, a single OS is loaded into a computing system. However, IBM zSeries servers can be partitioned into separate logical computing systems. System resources—memory, processors, and I/O devices—can be divided or shared among many such independent logical partitions (LPARs) under the control of an LPAR hypervisor built into the zSeries system [4]. Each LPAR supports an independent OS loaded by a separate IPL operation. Further, when the z/VM OS is running in an LPAR, the LPAR resources can be further divided or shared among a large number of independent virtual machines (VMs). An OS running in a VM is said to be a *guest* operating system of the z/VM operating system, and is referred to as a *VM guest*. Each LPAR and each virtual machine can use the SCSI IPL function.

The challenges

To understand the additional complexities introduced by SCSI IPL compared with CCW IPL, a general understanding of the latter is needed. As mentioned above, I/O devices used for CCW IPL are controlled by channel programs comprising a sequence of CCWs. A CCW contains a command to perform an individual read, write, or control operation for an I/O device. CCWs also designate the memory and storage areas associated with read and write operations. I/O operations are initiated by an OS on a central processing unit (CPU) and are scheduled for execution in a channel by a system process running on a system assist processor (SAP). The channel programs are executed by channel engines running independently of the CPUs and SAPs.

Each CCW-based I/O device is identified to an OS by a two-byte device number. The system programmer defines the I/O configuration, including the assignment of a device number to each I/O device, through the I/O configuration

0018-8646/04/$5.00 © 2004 IBM

program (IOCP) [5]. The IOCP creates an I/O configuration data set (IOCDS) that describes the configuration to the zSeries server, providing the correlation between the device number known to an OS and the physical device connected to a channel.

CCW IPL is initiated by the system operator using zSeries manual controls, such as those provided at the support element (SE) or a hardware management console (HMC). An SE supports a single zSeries server and provides the operator with a variety of controls to perform functions such as power-on and power-off, various types of resets, IPL, and many other functions. An HMC provides a subset of the same controls, but can communicate with more than one SE, and hence with more than one zSeries server at a time, allowing for consolidated operation of multiple servers. (For a VM guest, CCW IPL is initiated automatically at user logon or by a command at the user's terminal.) The system operator supplies only the device number of an I/O device on which resides a bootstrap-loader program for the OS to be loaded. This device becomes the IPL device. Upon initiation of an IPL operation, the system looks up the applicable channel, control-unit, and I/O-device identifiers for the specified IPL device number as described in the IOCDS. The system initiates a channel program that begins with an implied Read CCW, executed as if it existed at memory location 0, that reads the first 24 bytes (three doublewords) available from the IPL device into locations 0–23 of logical-partition memory. (For a virtual machine, "the system" is the z/VM OS rather than functionality built into the zSeries server itself, and the memory is the virtual memory of the virtual machine.)

For disk IPL devices, the first available bytes are in record 1 on track 0 of cylinder 0. The doubleword at location 0 typically contains a program status word (PSW) that is later used to begin instruction execution. Command chaining from the implicit Read CCW causes execution of the channel program to continue at location 8. Thus, the second doubleword is typically a Read CCW to read additional CCWs from the IPL device into memory. The third doubleword is typically a transfer-in-channel (TIC) CCW that branches to the additional CCWs. These CCWs typically begin reading the OS into memory. When the channel program completes, the contents of the doubleword at address 0 are made the current PSW. The IPL process is considered complete, and instruction processing begins in the OS bootstrap loader, which then completes the loading and initialization of the OS.

SCSI IPL is a more complex process. In addition to identifying the IPL device by a two-byte device number, SCSI devices are identified by two eight-byte numbers: a worldwide port name (WWPN) that identifies the Fibre Channel (FC) port of a SCSI *target device* in the FC fabric, and the logical unit number (LUN) that identifies the logical device, or *logical unit*, within the SCSI target device [6]. A logical unit serves as the zSeries IPL device.

In addition to requiring more parameters to identify the IPL device, the operational steps are more complex. First, a *login* operation with the FC fabric is required. An FC fabric is composed of switches that connect storage controllers in a storage area network (SAN), known as SCSI target devices, and host computing systems, known as SCSI initiator devices. The login process requires the transfer of the unique WWPN of the host channel to its adjacent switch in the fabric. In response, the switch provides a shorter (three-byte) identifier used to identify the host channel in subsequent communications within the fabric. The next step involves contacting a name server within the fabric to obtain the address of the SCSI target device to be accessed. The FCP channel performs a port login to the target device port, followed by a process login to initiate FCP operations. At this point data can be read from the logical unit serving as the IPL device.

The number and complexity of logical operations needed to perform I/O through an FC fabric is much greater than for reading 24 bytes from a directly attached CCW-type I/O device. A complex program is needed to construct SCSI Read commands, encapsulate them into FC command sequences, and extract data from the corresponding response sequences. This greater logical complexity of operations for SCSI IPL and the corresponding increase in the number of input parameters made it impractical to simply extend the implicit system operations of CCW IPL. Therefore, a new mechanism for SCSI IPL was developed.

The solution

SCSI IPL is performed by a combination of zSeries system functions and a special program called the machine loader. The machine loader executes in the program memory (LPAR or virtual machine) in which the IPL is being performed. As with CCW IPL, a system operator uses zSeries manual controls at the SE or an HMC to initiate a SCSI IPL. (Following the pattern of CCW IPL for a VM guest, SCSI IPL of a guest is initiated automatically at user logon or by a command at the user's terminal.) The SE goes beyond its function of handling operator input in CCW IPL, and places the machine loader and the IPL parameters in the LPAR memory.

An overview of the SCSI IPL process initiated at the SE for an LPAR is shown in **Figure 1**. The primary components are the SE, the machine loader (ML), the LPAR memory, the SCSI IPL device, and the OS loader. To initiate a SCSI IPL, the operator enters the IPL parameters as shown in Step 1. One way to do this is to enter them at the SE console as shown in **Figure 2**. Since there are several IPL parameters, some of which are multiple bytes in length, the parameters can be saved in

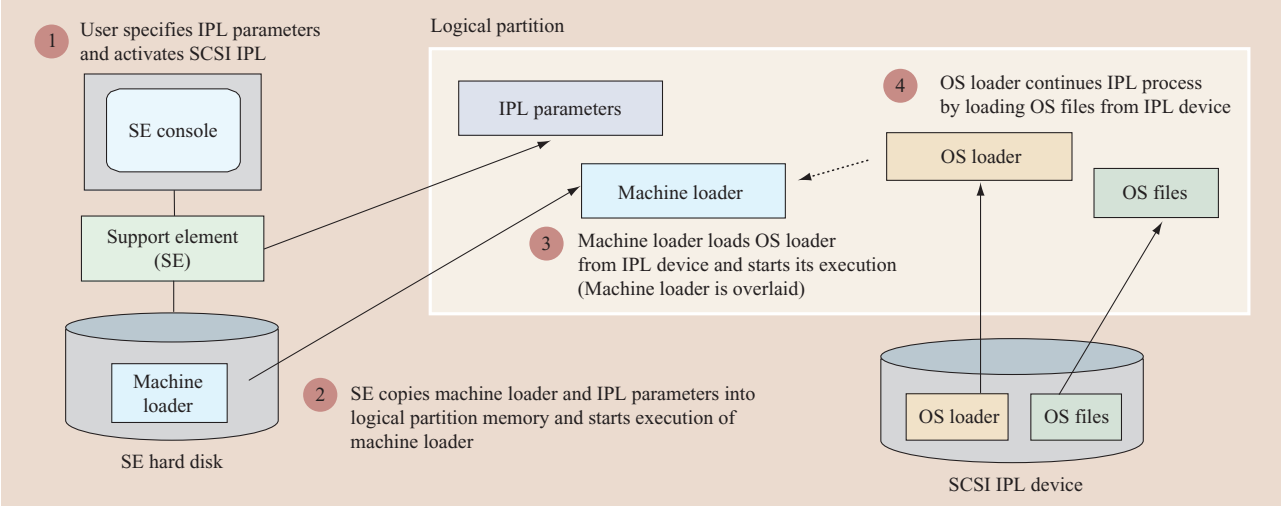

Figure 1

Control flow for a logical partition IPL.

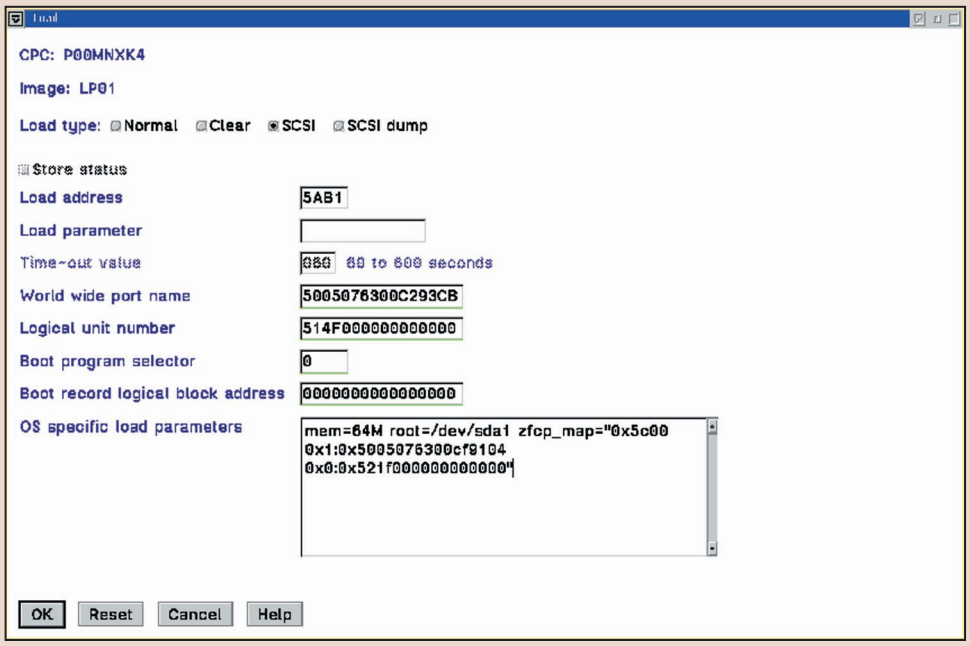

Figure 2

HMC/SE panel used to initiate a SCSI IPL.

activation profiles for easy reuse in subsequent IPL operations. After the parameters are entered, the operator starts the IPL by an entry at the console.

The SE begins the IPL by transferring the machine loader from SE local memory into LPAR memory as shown in Step 2 of Figure 1. The SE also stores the IPL

parameters in a specified location in LPAR memory in a specified format to be available for use by the machine loader. When this is complete, the SE signals the LPAR hypervisor to start the execution of the machine loader.

In Step 3, the machine loader executes in the LPAR memory with all of the available zSeries programming facilities. This allows the machine loader to perform the complex operations necessary to complete the SCSI IPL. First, the machine loader uses the IPL parameters stored by the SE in LPAR memory to locate the IPL device and the location on the IPL device of the program to be loaded. Next, SCSI commands are built to access the IPL device, and the machine loader determines the locations in LPAR memory in which to place the program. The SCSI I/O operations are initiated through a zSeries FCP channel to load the program. Note that the machine loader must avoid overwriting itself when loading the program, so the machine loader relocates itself when necessary during this process. Finally, control is transferred to the loaded program, completing the IPL operation. By this time, there is no further need for the machine loader, so it can be overwritten. Typically, the program to be loaded is an OS loader, which loads the rest of the OS from the IPL device as shown in Step 4.

The IPL concept described here goes beyond the boot capabilities available on most other computer systems. Typically, a relatively simple routine, such as the basic input/output system (BIOS), controls the initial OS boot process [7–9]. Such IPL routines usually support a very limited set of devices with fixed device formats, and since they are typically implemented in read-only storage, they are not easily modified. The process described in this paper, in contrast, supports the use of an arbitrarily complex IPL routine, referred to as the *machine loader*. In a typical non-zSeries machine, after power-on self-test (POST), the system BIOS identifies the boot device, typically an internally attached integrated drive electronics (IDE) or SCSI disk. The BIOS then reads the first physical block from the device, called the master boot record (MBR), and executes a piece of code contained in this block [7]. Some systems incorporate FC adapters that provide BIOS extensions, allowing use of an FC-attached device. However, these extensions are often restricted to point-to-point FC connections, and they often require an update to the BIOS to specify the address of the boot device.

The concepts used in zSeries SCSI IPL significantly enhance the capabilities of IPL, as described above. By using an arbitrarily complex machine loader and loading it into the machine from the SE to control the IPL process—instead of using a routine that is stored in read-only storage—the IPL routine can be upgraded more easily. Since the machine loader can be arbitrarily complex, it can fully utilize all of the capabilities of the machine, and it can support a wide range of boot devices at any address in an FC SAN. The zSeries SCSI IPL process also supports simultaneous execution in multiple logical partitions or z/VM guests. Details of how these new concepts were applied to SCSI IPL, along with a variation of SCSI IPL that dumps the contents of program memory to a SCSI disk, are provided below.

IPL of a logical partition

Initiating IPL

On a zSeries server, IPL of an LPAR can be initiated from an HMC or an SE in different ways. In particular, IPL can be initiated either manually via a specific load task or implicitly as part of a process called activation. The latter is a process that takes the LPAR into a desired state, such as the "IPL-complete" state, that may include prerequisite functions, such as powering-on the entire system and starting the LPAR, depending on the state of the system. Activation can be scheduled to occur at a predefined date and time. An activation profile is the sequence of steps and associated parameters that define the desired activation sequence. The profile includes the IPL parameters that are entered from the load panel of the load task when the IPL is done manually.

The load task and activation methods are both available from the SE or the HMC. In addition, either method can be initiated through a Web interface provided by the HMC. The actual IPL of an LPAR is always done by the SE. If IPL is initiated via the HMC, a special protocol is used to route the request to the SE. From this point, we describe only the actions performed by the SE.

Before initiating an IPL operation, the operator must choose the type of IPL and supply the IPL device information. There are two types of CCW IPL, called *load normal* and *load clear*. Similarly, there are two types of SCSI IPL, called *SCSI load with dump* (in which the contents of main storage are preserved, as with load normal), and *SCSI load* (in which the contents of main storage are cleared, as with load clear). All are available via both the load task and the activation methods. For CCW IPL, two IPL parameters apply: a load address (the device number of the IPL device) and an optional load parameter to be passed to the OS being loaded. Many more IPL parameters are required for SCSI IPL, all of which have been added to the load panel of the load task and to the activation profiles. Figure 2 is an example of a load panel with input values to select a SCSI IPL; it is discussed in detail below.

The IPL parameters are placed in the LPAR memory in the form of an extensible markup language (XML) string. The SE builds the XML string describing the type of IPL and all of the relevant parameters. The XML string is made available to the machine loader as well as the

```
<eServer_ipl_script version="1.0">

 <type>ipl</type>

 <ipl_control_section id="first_ipl">
  <ipl_platform_loader>
   <fcp_ipl>
    <devno>0x5AB1</devno>
    <wwpn>0x5005076300C293CB</wwpn>
    <lun>0x514F000000000000</lun>
    <br_lba>0x0000000000000000</br_lba>
    <boot_program_selector>0x00000000</boot_program_selector>
   </fcp_ipl>
  </ipl_platform_loader>
  <system_control_program>
   <parameter_string>
    mem=64M root=/dev/sda1 zfcp="0x5c00 0x1:0x50050076300cf9104
0x0:0x521f000000000000"
   </parameter_string>
  </system_control_program>
 </ipl_control_section>

</eServer_ipl_script>
```

Figure 3

SCSI IPL parameters shown as an XML string.

program being loaded. **Figure 3** shows the XML string created from the load panel input of Figure 2.

When the IPL operation is complete, feedback about success or failure and any error messages are shown on the SE or the HMC console from which the IPL operation was initiated. The SE regards the IPL as complete when the machine loader is successfully loaded into an LPAR and started. The machine loader behaves like any OS in an LPAR, reporting its progress and potential error messages via a special operating-system messages console that can be made visible on the SE or an HMC.

Input parameters

The panel shown in Figure 2 allows the selection of different types of IPL. Load types normal (load normal) and clear (load clear) are used to initiate CCW IPL. These two types differ in the reset operations performed in the context of the IPL. A single load address is required, and a single eight-byte load parameter can be optionally specified.

Load type SCSI (SCSI load), as selected in the example in Figure 2, is used to initiate SCSI IPL of an OS, and an extended set of parameters can be specified. The same parameters are valid for load type SCSI dump (SCSI load with dump), which can be used to initiate SCSI IPL of a dump program, as described later.

For CCW IPL, the load address specifies the two-byte device number (four hexadecimal digits) of the IPL device. For SCSI IPL, the load address specifies a two-byte device number associated with the FCP channel to be used to access the FC fabric in which the IPL device resides. This device number, defined in association with the channel in the IOCDS, is not associated with a real I/O device, but rather with a pair of queues in the channel for exchanging commands and responses between the program (the machine loader in this case) and the channel [3].

The *load parameter* specifies an optional eight-character extended binary-coded decimal interchange code (EBCDIC)-encoded load parameter to be passed to the OS being loaded.

The *time-out value* is used only for CCW IPL. Some devices take longer to respond to an IPL request, and the time-out value allows the user to specify a longer time-out than the default value.

The *worldwide port name* is the eight-byte permanent name (16 hexadecimal digits) assigned to the FC adapter port of the SCSI target device containing the logical unit serving as the IPL device. The FC fabric must be configured in such a way that the FCP channel used for the IPL operation has access to this port.

The *logical unit number* (or LUN) is the eight-byte identifier (16 hexadecimal digits) of the logical unit representing the IPL device.

The *boot program selector*, a decimal value between 0 and 30, is used to select the section of the IPL disk on which the desired OS resides. (The SCSI IPL function allows up to 31 different operating systems to reside on one IPL disk.)

The *boot record logical block address* specifies the logical block address (LBA) of the boot record. (A boot record is used to locate an OS loader on an IPL disk. Normally,

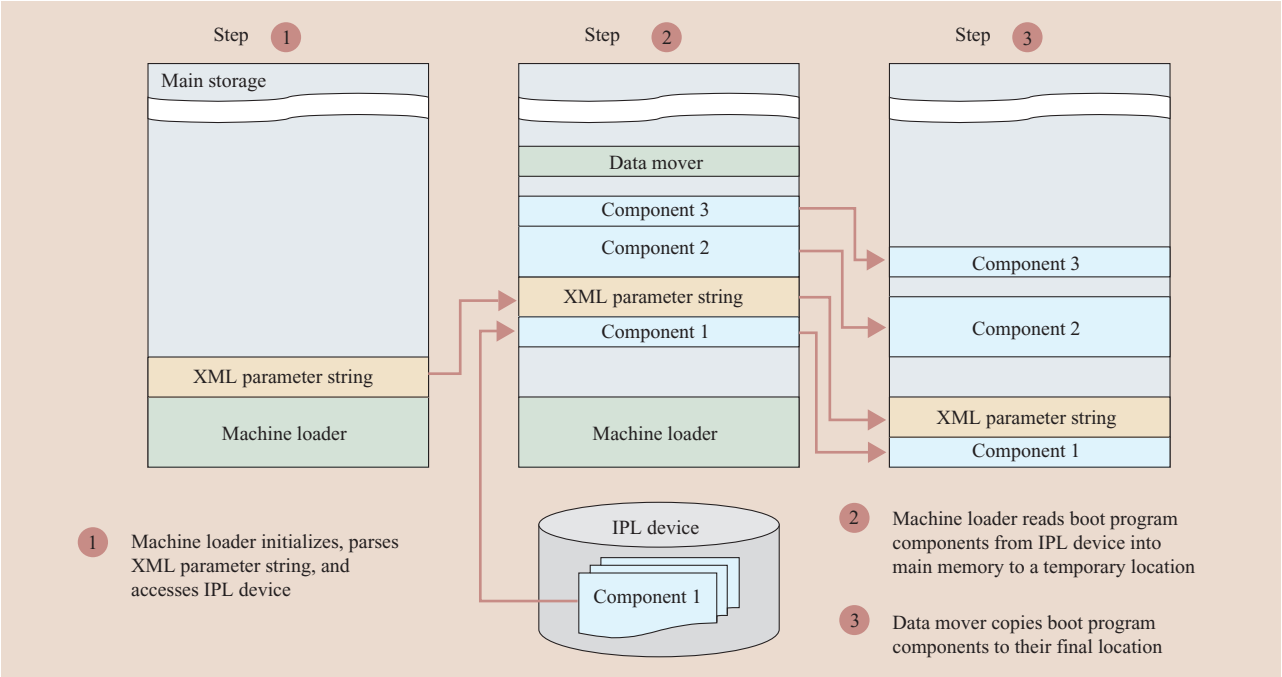

Figure 4

Operation of the machine loader.

this boot record is located at LBA 0. The SCSI IPL function allows the boot record to be located at a different LBA.)

Finally, the *OS-specific load parameter* is optionally used to pass a string of characters to the program being loaded. Neither the system nor the machine loader interprets or uses these parameters. For example, the OS-specific load parameter can be used to identify additional I/O devices and related storage addresses required by the OS being loaded.

The same IPL parameters apply to SCSI dump. The difference from SCSI load is that the IPL device must contain a dump program rather than an OS. A dump program for an OS is used when the OS has failed and it is necessary to dump the contents of memory to a storage device for analysis.

Loading and starting the machine loader

After the SE obtains and checks the IPL parameters from the load panel or an activation profile, the SE builds the XML string from the parameters. The SE requests the LPAR hypervisor to perform a clear reset of the LPAR to be loaded, much as for a load clear CCW IPL. When the reset is complete, the SE reads the machine loader from the SE local disk and moves it into the LPAR memory starting at address 0. This is done using a generic service-word interface that is provided for communication between the SE and the system. The SE also moves the XML string into the LPAR memory at an architecturally defined address not occupied by the machine loader.

Next, the SE requests the LPAR hypervisor to start execution of the machine loader. The LPAR hypervisor executes a Restart PSW function, which sets the current PSW from the contents of locations 0–7 of the LPAR memory (i.e., the first eight bytes of the machine loader). This PSW contains the address of an entry point in the machine loader, and instruction processing begins at this address.

Operation of the machine loader

The operation of the machine loader is diagrammed in **Figure 4**. The machine loader begins with some initialization, including the allocation of memory and possible relocation of the XML string to a different memory area. Next, the machine loader parses the XML string, checking for correctness and validity. If problems are detected, the machine loader displays appropriate error messages at the operating-system message console and terminates its operation.

When parsing is successful and all of the necessary information has been obtained, the machine loader begins I/O operations to the IPL device. First, the machine loader establishes communication with the FCP channel

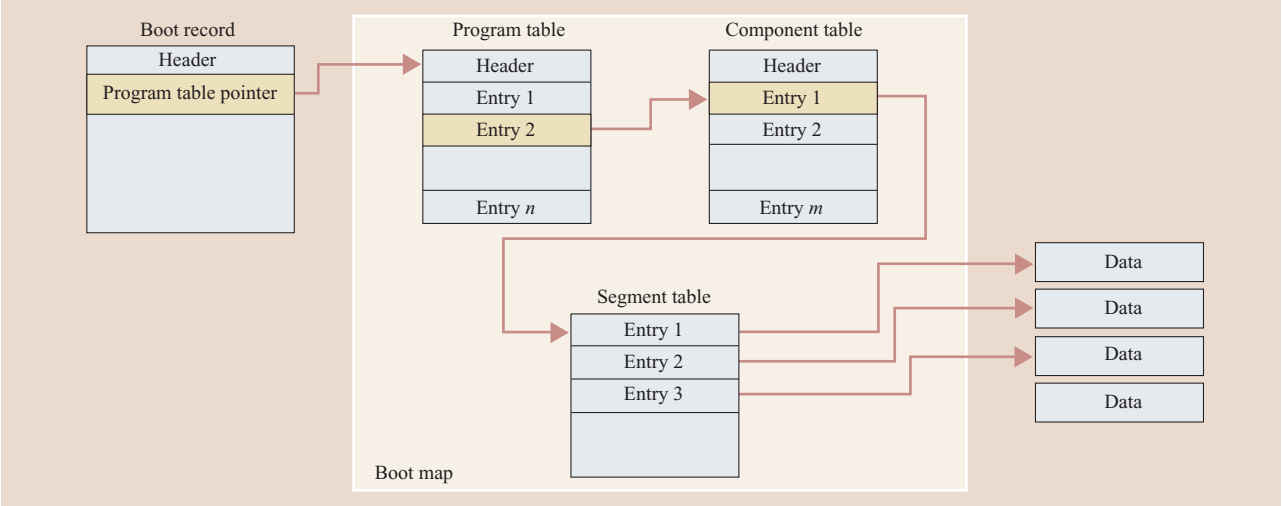

Figure 5

Layout of the data on the IPL disk.

using the specified device number. Next, the machine loader issues a request to the fabric name server with the WWPN for the SCSI target device containing the logical unit to be used for the IPL. The FC name server resolves this WWPN into a corresponding three-byte identifier for use by the FCP channel in communications with the SCSI target device. Using this FC identifier, the machine loader issues an Open Port request for the target port to the FCP channel [3], which in turn performs an N_Port Login function for this port [10] on the target device, if required. Finally, the machine loader uses the LUN from the input parameters to issue an Open LUN request to the FCP channel [3], which establishes a logical connection in the channel to the logical unit serving as the IPL device.

If these steps are completed successfully, the machine loader issues SCSI Read requests to the IPL device to determine whether the IPL device has a valid IPL format. The overall layout of the data on the IPL device is shown in **Figure 5**. First, the machine loader looks for a boot record at the specified LBA. Normally, this boot record is expected to be at LBA 0, but the operator may specify a different LBA. If a valid boot record is found at the specified LBA, the machine loader checks an indicator in the boot record header that indicates that the boot record is in the zSeries format. If the format is zSeries, the machine loader extracts a program table pointer (LBA) from the boot record.

The program table addressed by this pointer contains the zSeries format indicator and LBAs of additional control structures for up to 31 bootable programs on the IPL disk. The boot program selector from the input parameters identifies a particular program table entry, which represents a particular OS.

At this point, the machine loader has used all of the required IPL parameters, and the IPL device itself contains all of the additional information to complete the IPL. The program table entry contains the address of a component table specific to the OS being loaded. An OS comprises one or more components to be loaded into different locations in the LPAR memory, and each component is described in a component table entry. The component table header contains the zSeries format indicator, plus another indicator that the program is an OS (for a SCSI load operation) or a dump program (for a SCSI dump operation).

Each component table entry defines the type of component and the address in LPAR memory where the component is to be loaded. While each component is loaded into a contiguous area in memory, the code segments comprising the component may be scattered in several segments on the IPL device; therefore, the component table entry for each component contains the address of a segment table. Each segment table entry contains the address (LBA) and size of a segment of the OS being loaded.

The machine loader sequentially processes components and segments within components. After validating the information in the data structures, the machine loader reads the program code and data associated with the selected OS (addressed by the segment table entries) into the LPAR memory (see Step 2 in Figure 4). However, in most cases the machine loader is not able to store the

data in the target memory locations specified in the data structures mentioned above, since this data would then overwrite the memory area occupied by the machine loader or the XML string. Instead, the operating-system segments are placed in temporary buffers with addresses higher than the memory locations occupied by the machine loader itself (see Figure 4).

While loading the different components into main storage, the machine loader creates a table called a *move map*. The move map contains the current (temporary) component location, the component size, and the ultimate target location in the LPAR memory. The machine loader might once again be required to relocate the XML string. If an examination of the component target location in memory reveals that a component would overwrite the XML string, the XML string is moved to another area where no conflict exists. This is done by creating an entry in the move map for the XML string.

With the OS components all loaded into memory, the machine loader moves the components and the XML string to their final locations. This cannot be done by the mainline machine loader code, since this process would, in most cases, overwrite the machine loader itself. Therefore, the machine loader includes a small *move routine* capable of performing this move operation. The move routine is copied to a memory location above all of the areas that currently contain needed program code or data or will contain it after the relocation (see Step 3 in Figure 4). The machine loader branches to the move routine to perform the moves according to the move map. This process typically overwrites the machine loader, which is no longer needed.

The move routine ensures that the doubleword at memory address 0 contains a PSW by which the OS can be started by the LPAR hypervisor. There are two ways for the starting PSW to be specified on the IPL device. One method is to include the starting PSW in the first eight bytes of a component to be loaded at memory location 0. Alternatively, the starting PSW can be specified explicitly in the last component table entry. In this case the move routine stores the specified PSW into LPAR memory locations 0–7.

The final task of the move routine is to store the address of the XML string in an architecturally defined location for use by the OS loader and issue a special instruction to signal completion to the LPAR hypervisor. The `DIAGNOSE` instruction requests the LPAR hypervisor to perform a clear reset operation for the LPAR (except that the LPAR memory is not cleared) and to begin instruction processing for the OS loader by setting the current LPAR PSW from the contents of the doubleword at address 0.

IPL of a VM guest

The IPL of a VM guest from a SCSI IPL device is very similar to SCSI IPL in an LPAR (**Figure 6**). In particular, the same machine loader is used, performing the same functions. The VM OS or VM hypervisor performs a reset for the virtual machine and uses a special instruction to request that the system place the machine loader in the pages (four-kilobyte blocks) of the virtual machine memory. The system moves the machine loader from a reserved portion of the zSeries hardware system area (HSA) into the real page frames comprising the virtual machine memory. The VM hypervisor creates the XML string containing IPL parameters based on input received from the VM user or established by the VM administrator. The XML string is placed in the virtual machine memory, and the address of the XML string is placed in the architecturally defined location.

As in the case of an LPAR, the VM hypervisor starts execution of the machine loader by setting the current PSW of the virtual machine from the contents of bytes 0–7 of the virtual machine memory. The machine loader runs normally, with assistance from the VM hypervisor only when needed. For example, I/O instructions are intercepted by the VM hypervisor so that the virtual memory addresses used by the machine loader (or any other program in a virtual machine) can be translated into the real memory addresses used by the channel. The machine loader is not aware of such assistance because zSeries instruction interception gives control to a hypervisor (LPAR or VM) when required by the architecture or requested by the hypervisor, and the hypervisor causes the instruction to appear to have been completed in the architecturally defined manner. Typically, the hypervisor issues the same machine instruction that was issued by the guest, preceded and followed by any other required processing, as, for example, with the address translation described above for I/O instructions intercepted by the VM hypervisor.

The IPL of a VM guest is initiated from the guest operating-system console, i.e., the terminal at which the VM user logs on (see Step 2 in Figure 6). The same set of IPL parameters (i.e., load address, WWPN, LUN, etc.) are specified. The z/VM allows the establishment of the IPL parameters in advance (similar to an LPAR activation profile) by including them in the guest directory entry. The VM directory defines the identities of each virtual machine, the virtual resources available, and the real-system resources to be used to satisfy the virtual resource allocation. The SCSI IPL parameters are specified in a new `LOADDEV` directory statement. Similarly, there is a new user command, `SET LOADDEV`, by which the user can specify one or more of the IPL parameters. Any parameters not specified in the `SET LOADDEV` command remain as previously established in the directory or by an

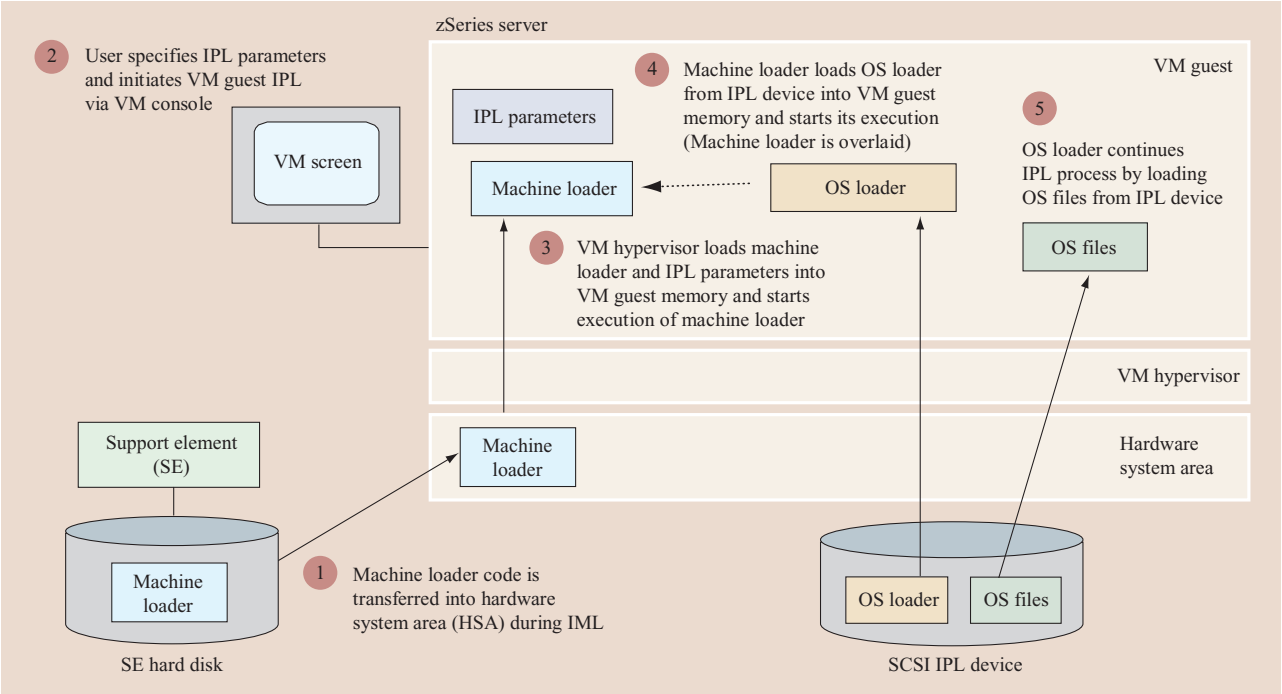

Figure 6

Control flow for a VM guest IPL.

earlier `SET LOADDEV` command. The `SET LOADDEV` command can also be used to clear previously established SCSI IPL parameters. When all required SCSI IPL parameters have been established, the user can issue the existing IPL command to begin the loading and starting of the virtual machine. The z/VM retains the IPL parameters after IPL such that, if the same parameters are to be used for the next IPL, the user need only enter the IPL command again.

A difference between SCSI IPL of a VM guest and an LPAR is that, for an LPAR, the machine loader is transferred directly from the SE into the memory of the LPAR each time an LPAR is loaded. As a performance accommodation to z/VM, the zSeries system places a copy of the machine loader in HSA when the system undergoes initial microprogram loading (IML) (Step 1 in Figure 6). This avoids the possibility of repeated, lengthy transfers across the SE-to-system interface as multiple virtual machines are loaded. Instead, the placement of the machine loader in the memory of a virtual machine occurs at memory-to-memory speed (Step 3 in Figure 6). Many virtual machines can be loaded in parallel without delaying each other, as they would if each virtual machine had to receive the machine loader directly from the SE.

The special `DIAGNOSE` instruction issued by the machine loader to complete its operation is intercepted by the VM hypervisor instead of the LPAR hypervisor. The VM hypervisor performs the required reset of the virtual machine and begins instruction processing for the guest OS loader by setting the current PSW of the virtual machine from bytes 0–7 of the virtual memory. After that, the IPL process continues much in the same way as described above for an LPAR IPL (Steps 4 and 5 in Figure 6).

The VM hypervisor itself can be a guest of another VM hypervisor in a recursive manner. The SCSI IPL function has been implemented in such a way that it works at any level of VM hypervisor nesting.

The z/VM does not support the SCSI load-with-dump function for guests, since z/VM already contains facilities outside the zSeries architecture by which the memory of a virtual machine can be dumped to devices available to the VM hypervisor.

IPL of a dump program

SCSI IPL can be used to load any kind of program that can run on its own, without the services of a separate OS. Typically, such a program is an OS, but it can also be a test platform or what is known as a *standalone utility* (a utility that runs without a separate OS because it "stands alone" by providing all of its own needed services). Special provisions are made in SCSI IPL for standalone dump

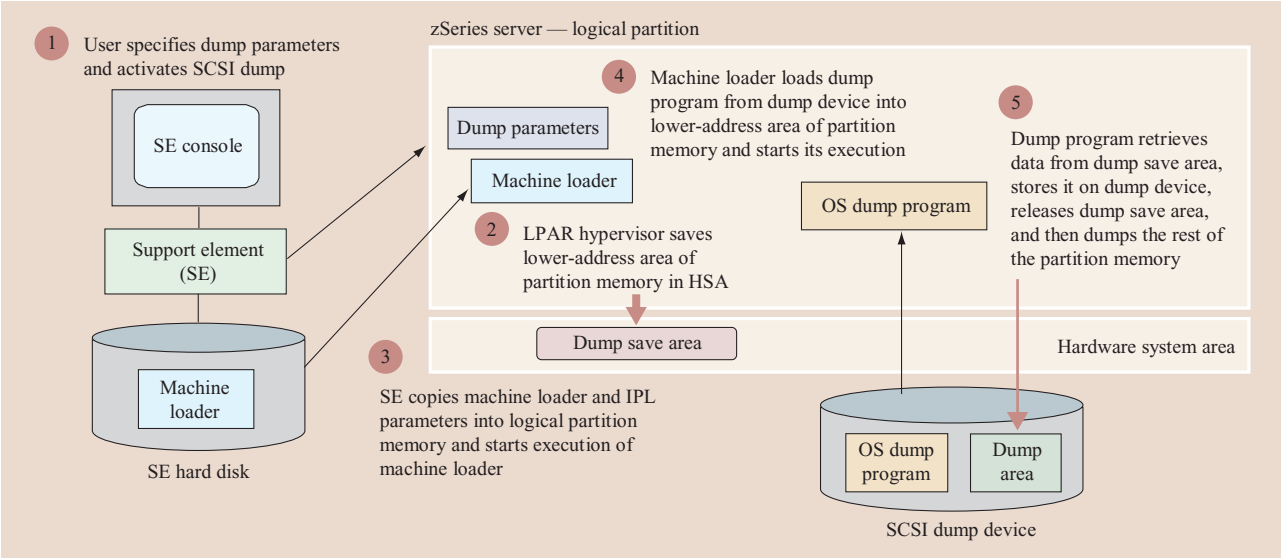

Figure 7

Control flow for a standalone dump operation.

programs, which are utility programs to dump the contents of program memory to, in this case, a SCSI disk. It is these programs for which the SCSI dump form of IPL is provided.

Dump programs create a special challenge for SCSI IPL. The function of the dump program is to save the entire contents of program memory on an I/O device. Thus, it is not acceptable to simply place the machine loader in this same memory, because that would overwrite data that should be saved.

When load type *SCSI dump* is selected to load a dump program into the memory of an LPAR (Step 1 in **Figure 7**), rather than send a *clear reset* request to the LPAR hypervisor to clear the LPAR memory, the SE sends a SCSI dump request before placing the machine loader in the LPAR memory. The LPAR hypervisor handles the SCSI dump request by performing a *store status* operation for the logical CPU being IPLed in the LPAR, performing a *reset normal* operation for the LPAR, and copying the contents of the lower-address area of the LPAR memory into a reserved area of HSA (Step 2 in Figure 7). The amount of LPAR memory area that is saved is large enough that nothing is lost from it when the machine loader, the dump program loaded by the machine loader, and their execution environments are placed there. If memory-access problems prevent any particular memory page from being saved in HSA, the LPAR hypervisor marks the save area for that page, indicating that the contents of the HSA page are not valid and are thus not available to be retrieved by the dump program, as described below.

To avoid allocating excessive HSA for this purpose, one save area exists for all LPARs. Use of the save area is serialized by the system to prevent simultaneous use by more than one LPAR. Thus, saved data from one LPAR cannot be intermixed inadvertently with that of another LPAR, and no LPAR can gain access to the saved data of another LPAR.

After saving the necessary amount of LPAR memory, the LPAR hypervisor signals the SE to place the machine loader in the LPAR memory (Step 3 in Figure 7). When this is complete, the SE signals the LPAR hypervisor to begin execution of the machine loader. The machine loader runs as it does for a regular SCSI IPL, except that the program on the IPL disk must be marked as a dump program in the component table header, as mentioned above (Step 4 in Figure 7). Also, when the machine loader completes its operation, it uses a different form of the `DIAGNOSE` instruction to pass control to the LPAR hypervisor. This form of the `DIAGNOSE` instruction signals the hypervisor to perform a normal reset instead of a clear reset operation before setting the current LPAR PSW from the contents of the LPAR memory locations 0–7. The effect of the normal reset is to preserve the state of any additional logical CPUs in the LPAR, allowing the dump program to retrieve their state. The saved data in HSA is also preserved for access by the dump program.

After the dump program is loaded, it determines where to put the dump data. The XML string given to the dump

program identifies the device from which the dump program itself is loaded. There are different ways for this dump program to determine where to put the dump data. One method is to format the device where the dump program resides so that the dump data can be stored on the same device. This is the method chosen by the current implementation of Linux for zSeries. Another possibility is to pass a different device address via the load parameter (the optional IPL parameter passed through to the program being loaded). Once the dump program has identified the dump device, the dump program retrieves the data that was saved in HSA. A new instruction interface is provided for the dump program to retrieve this data. After all saved data is retrieved and stored on the dump device, the dump program signals the LPAR hypervisor to release the HSA save area, enabling its reuse in a subsequent dump process for other LPARs.

Because the dump program is responsible for signaling the LPAR hypervisor to release the HSA save area, the machine loader checks that the program being loaded is indeed a dump program when a SCSI dump is requested. As mentioned earlier, the IPL disk indicates in the component table header whether the program is an OS or a dump program. The machine loader checks whether the value of this indicator matches the operation requested from the SE. The machine loader aborts the process with an error indication in case of a mismatch. After the release of the HSA save area, the dump program continues dumping the LPAR memory to the dump device until the dump is complete (Step 5 in Figure 7).

Conclusions

In this paper we described the new SCSI IPL function of IBM zSeries servers and described how it employs several new concepts not typically used during IPL procedures. These new concepts allow IPL of operating systems from SAN-attached SCSI disks into the LPARs of zSeries servers and virtual machines instantiated by z/VM on these servers. A variation of SCSI IPL enables the contents of LPAR memory to be dumped onto a SCSI disk.

With this new feature, the Linux for zSeries OS can be installed on SCSI disks connected to zSeries servers via Fibre Channel protocol channels and a Fibre Channel fabric. SCSI IPL can be exploited by other operating systems in the future as they add support for SCSI I/O. The function introduces the concept of using a complex, full-featured machine loader program that is built into the zSeries server via its support element and placed in the program memory of an LPAR or virtual machine to perform the SCSI IPL. This concept permits full exploitation of z/Architecture* facilities when retrieving an OS loader or a dump program from something as complex as an FC fabric. The machine loader can be extended in the future to support IPL and dump operations involving new I/O protocols, fabrics, and devices as new I/O technologies develop.

Acknowledgments

The authors thank all of our colleagues who supported this project during its design and execution. Our special thanks go to Ingo Adlung, Chuck Brazie, Heiko Carstens, Dan Clarke, Wolfgang Eckert, Joe Goldsmith, Clay Harshberger, Michael Holzheu, Elisabeth Kon, Steve Kriese, Juergen Leopold, Jim McGinniss, Stefan Mueller, Frank Novak, Michel Raicher, Christoph Raisch, Otto Ruoss, Albert Schirmer, Raimund Schroeder, Holger Smolinski, Khadija Souissi, Rick Tarcza, Pamela Walford, Steve Wilkins, and all the others who helped to make this project a success. We also thank the anonymous referees for their time spent reviewing this paper and for their valuable comments.

References

1. IBM Corporation, *Enterprise Systems Architecture/390 Principles of Operation* (SA22-7201); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
2. IBM Corporation, *z/Architecture Principles of Operation* (SA22-7832); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
3. I. Adlung, G. Banzhaf, W. Eckert, G. Kuch, S. Mueller, and C. Raisch, "FCP for the IBM eServer zSeries Systems: Access to Distributed Storage," *IBM J. Res. & Dev.* **46**, No. 4/5, 487–502 (July/September 2002).
4. IBM Corporation, *IBM eServer zSeries z990 System Overview* (SA22-1032); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
5. IBM Corporation, *IBM eServer zSeries Input/Output Configuration Program User's Guide for ICP IOCP* (SB10-7037); see *http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi/*.
6. ANSI/INCITS, Technical Committee T10, "Information Systems–Fibre Channel Protocol for SCSI, Second Version (FCP-2)," American National Standards Institute and InterNational Committee for Information Standards, Washington, DC, 2001.
7. "The Master Boot Record (MBR) and Why Is It Necessary?"; see *http://www.dewassoc.com/kbase/index.html*.
8. R. Brown and J. Kyle, *PC Interrupts, A Programmer's Reference to BIOS, DOS, and Third-Party Calls*, Addison-Wesley Publishing Company, Reading, MA, 1994.
9. D. A. Solomon and M. E. Russinovich, *Inside Microsoft Windows 2000*, Third Edition, Microsoft Press, Redmond, WA, 2000.
10. ANSI/INCITS, Technical Committee T11, "Fibre Channel Framing and Signaling (FC-FS)", American National Standards Institute and InterNational Committee for Information Standards, Washington, DC, 2002.

Received September 22, 2003; accepted for publication November 18, 2003; Internet publication April 6, 2004

Gerhard Banzhaf *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (banzhaf@de.ibm.com).* Dr. Banzhaf received his M.S. degree in computer science from the University of Karlsruhe (TH) and his Ph.D. degree in electrical engineering from the University of Siegen (GHS). In 1981 he joined the IBM Development Laboratory at Boeblingen, where he was involved primarily with the development of I/O subsystems and microcode for S/390 and zSeries systems. In particular, his focus was on local-area networks, infrastructures for the attachment of industry-standard I/O, integrated storage, and the attachment of SCSI Fibre Channel (FCP) devices.

Frank W. Brice *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (fbrice@us.ibm.com).* Mr. Brice is a Senior Technical Staff Member in z/VM Development, performing I/O-architecture and system-design work for z/VM and the zSeries platform. He received a B.S. degree with honors from the Stevens Institute of Technology in 1968, after which he joined the General Electric Company as a radar-systems engineer. Taking an educational leave of absence in 1971, he received an M.S. degree in computer science from Rutgers University in 1972. Returning to GE, he joined the Space Division to work on software for the NIMBUS and other satellite-imaging programs. Mr. Brice joined IBM in 1974 in Kingston, New York, working initially in the software-assurance organization. He joined VTAM Development in 1976, leading the physical-unit services team and later representing VTAM on the Systems Network Architecture Maintenance Board. He worked briefly in service-processor software development in 1983 and 1984 and then joined VM Development as an I/O-software designer. He developed expertise in I/O architecture and virtualization and continues working in these areas today. Mr. Brice is a co-inventor on 13 U.S. patents, with several additional patents pending; he has received several IBM awards.

Giles R. Frazier *IBM Systems and Technology Group, 11400 Burnet Road, Austin, Texas 78758 (grf@us.ibm.com).* Mr. Frazier is a Senior Technical Staff Member in the eServer I/O Architecture Department. His recent activities include work in the development of the IBM FICON Architecture, for which he received two IBM Outstanding Innovation Awards, development of various aspects of SCSI architecture related to mapping the SCSI protocol over Fibre Channel (FCP), and participation in related design teams. Over the past ten years he has been active in industry standards groups including the InfiniBand Trade Association and the Fibre Channel Industry Association, and was editor of the FC-SB-2 and FC-SB-3 standards. He received B.S. and M.S. degrees in electrical engineering from Stanford University and has been with IBM since 1973. Mr. Frazier is an inventor on seven patents with several more pending, and he has authored numerous publications. He is a registered Professional Engineer in the state of Texas.

Jeffrey P. Kubala *IBM Systems and Technology Group, 2455 South Road, Poughkeepsie, New York 12601 (kubes@us.ibm.com).* Mr. Kubala is a Senior Technical Staff Member on the z/OS Core Technical Design Team; he is currently the technical team leader for the zSeries LPAR hypervisor. He received a B.S.E. degree in computer engineering from the University of Connecticut and joined IBM in 1981. Since then, he has worked on compiler design and development, OS/390 Hiperbatch, and S/390 and zSeries logical partitioning. In addition to his role as the zSeries LPAR hypervisor technical team leader, Mr. Kubala is actively engaged with the iSeries and pSeries hypervisor teams as a technical consultant.

Thomas B. Mathias *IBM Systems and Technology Group, 1701 North Street, Endicott, New York 13760 (mathiast@us.ibm.com).* Mr. Mathias is a Senior Engineer in the IBM Systems Group. He received a B.S. degree in electrical engineering from Ohio State University and joined IBM in Endicott, New York, in 1984. He has worked in zSeries hardware development, and later in support element and hardware management console microcode development. Mr. Mathias was licensed as a Professional Engineer in the state of New York in 1992. He is a co-inventor on three U.S. patents and one pending patent application. Mr. Mathias has received three IBM Outstanding Technical Achievement Awards and numerous other IBM formal and informal awards.

Volker Sameske *IBM Systems and Technology Group, IBM Deutschland Entwicklung GmbH, Schoenaicherstrasse 220, 71032 Boeblingen, Germany (sameske@de.ibm.com).* Mr. Sameske is a software engineer in the IBM Systems Group. He received an M.S. degree in mathematics from the Technische Universität Bergakademie Freiberg and joined IBM in Boeblingen in 1999. Mr. Sameske has worked in zSeries VSE software development and service, and has been with the Linux for zSeries Development Team almost from the beginning of the project.