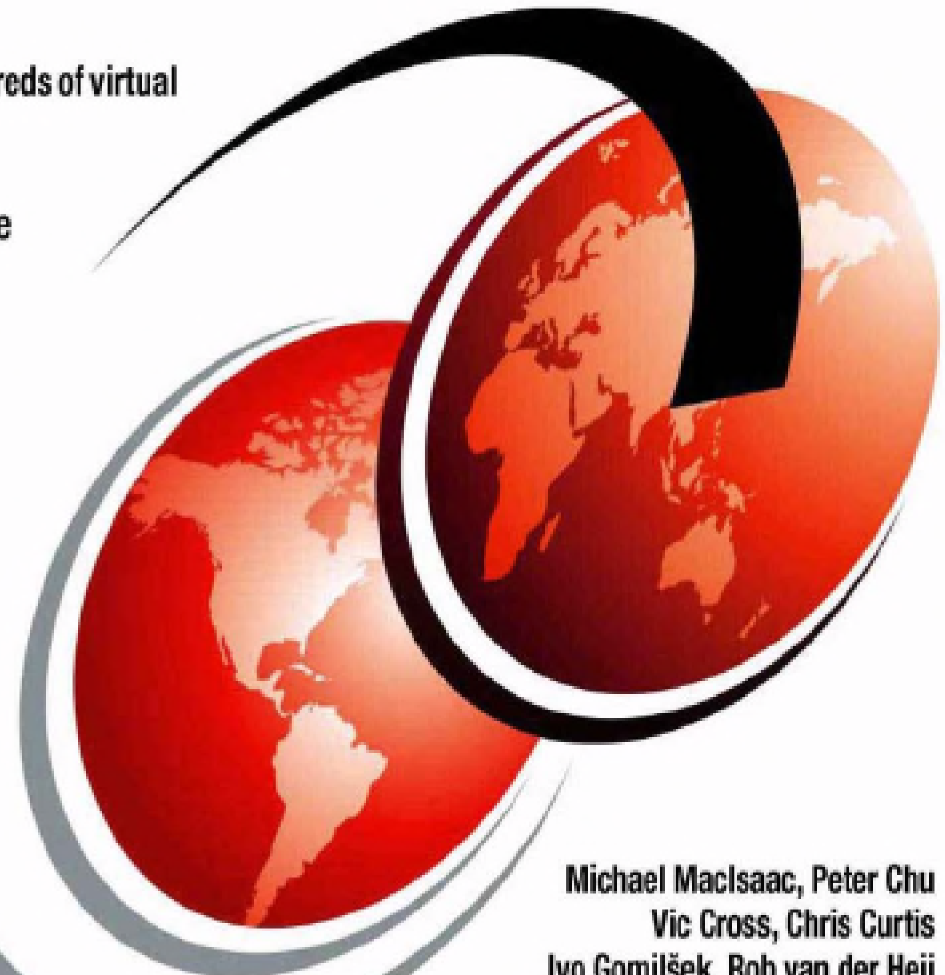
IBM

Linux on IBM @server zSeries and S/390: ISP/ASP Solutions



Create and maintain hundreds of virtual Linux images

Running on the mainframe

Managed by z/VM

Michael MacIsaac, Peter Chu
Vic Cross, Chris Curtis
Ivo Gomilšek, Rob van der Heij
Liz Holland, Barton Robinson
Martin Söllig, Simon Williams

Redbooks

ibm.com/redbooks

International Technical Support Organization

Linux on IBM @server zSeries and S/390: ISP/ASP Solutions

December 2001

SG24-6299-00

Take Note! Before using this information and the product it supports, be sure to read the general information in "Special notices" on page 455.

First Edition (December 2001)

This edition applies to z/VM 4.2 (ESP) and many different Linux distributions, but largely an internal development version of SuSE Linux Enterprise Server was used.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Contents

Preface

Now in IBM, we are committed to embracing Linux across everything that we do. Linux runs on our Intel servers, on our Power-based servers, on our iSeries. It runs on our mainframes, on our OEM technology, on our storage servers.

Linux, as we know, is the only operating system of which you can safely say: It will run on architectures that have not yet been invented.

- Dr. Irving Wladawsky-Berger

This IBM Redbook describes how Linux can be combined with z/VM on IBM @server zSeries and S/390 hardware - the mainframe. This combination of hardware and operating systems enables Internet Service Providers (ISPs) and Application Service providers (ASPs) to more efficiently provide services. (We assume a broad definition of ASP, to include production enterprise solutions as simple as file serving.)

When a new resource is required in a world of discrete servers, you can either add the workload to an *existing* server, or add another server to handle the workload. The less costly approach of adding the workload to an existing server often proves to be infeasible because the server lacks adequate capacity—or perhaps because you want to keep only one application on an existing server. As a result, frequently a new server is installed and a *server farm* thus begins to grow in an enterprise.

S/390 and zSeries hardware, microcode and software (especially PR/SM and z/VM) allow physical resources to be made virtual among Linux systems. This allows many hundreds of Linux systems to exist on a single server. Running multiple Linux *images* as guests of VM/ESA or z/VM is a smart choice. Consider the following benefits VM offers a Linux guest environment:

- Physical resources can be shared among multiple Linux images running on the same VM system. These resources include CPU cycles, memory, storage devices, and network adapters.
- Consolidating servers by running many Linux images on a single S/390 or zSeries offers savings in space, power consumption and administrative staffing.
- The virtual machine environment is flexible and adaptable. New Linux images can be added to a VM system quickly and easily without requiring dedicated resources. This also allows for a flexible test environment.

- The Linux images are able to take advantage of the hardware's reliability, availability and serviceability (RAS) features.
- VM allows high-speed communication among the Linux images, as much of the networking infrastructure can be virtual and thus performed in memory.
- VM's minidisk cache and virtual disks allow data-in-memory performance boosts.
- VM offers a rich debug environment that can be particularly valuable for diagnosing problems among the Linux images.
- VM's heritage of support for scheduling, automation, performance monitoring and reporting, accounting information and virtual machine management is available for Linux virtual machines as well.
- An effective way to grow the Linux workload capacity is either to add more Linux guests to a VM system (horizontal growth) or to simply raise the resources available to a Linux virtual machine (vertical growth).

This redbook is divided into two parts. The first part consists of theoretical discussions about installing and managing z/VM and Linux for zSeries and S/390 systems. The second part contains explicit examples of the work we did during the residency to create this redbook.

In our approach, we metaphorically refer to a collection of Linux servers running under VM as a "penguin colony". Like a colony of penguins, our virtual servers are individuals with their own attributes, sharing resources with neighbors. Some perform better than others, some provide services to other members of the colony, and some try to consume more resources than others. Penguins will be born, live, and (sometimes) die—in our environment, all of these events must be managed.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Michael MacIsaac is a team leader for S/390 redbooks and workshops at the ITSO Poughkeepsie Center. He writes about and teaches classes on Linux for S/390 and zSeries. Michael has worked at IBM for 14 years, mainly as a UNIX programmer. He has led teams that have written Redbooks on OS/390 UNIX, S/390 file and print serving, and Linux.

Peter Chu is an IBM zSeries Techline Specialist in California. He holds a degree in Computer Science from George Mason University. His areas of expertise include Capacity Planning for the zSeries and Linux.

Vic Cross is Linux for S/390 and zSeries Team Leader in the Professional Services division of Independent Systems Integrators, an IBM Large Systems Business Partner in Australia. He has 15 years of experience in general computing, six of which was spent working on S/390. He holds a Bachelor of Computing Science degree from Queensland University of Technology. His areas of expertise include networking and Linux.

Chris Curtis is a Senior Consultant with Satel Corporation, an IBM Business Partner in Salt Lake City, Utah, specializing in security consulting and providing managed security services. He has 12 years of experience in software and systems engineering on platforms of all sizes, from embedded through mainframe. He holds a degree in Computer Science from Westminster College. His areas of expertise include high-availability systems, databases, and advanced software architectures.

Ivo Gomilšek is an IT Specialist for Storage Area Networks, Storage and Linux in IBM Global Services - Slovenia for the CEE region. His areas of expertise include Storage Area Networks (SAN), Storage, IBM eServers xSeries servers, network operating systems (Linux, MS Windows, OS/2), and Lotus Domino servers. He is an IBM @server Certified Specialist in xSeries, a Red Hat Certified Engineer, and an OS/2 Warp Certified Engineer. Ivo was a member of the team that wrote the IBM Redbook *Designing an IBM Storage Area Network*, SG24-5758, and contributed to various eServer xSeries and Linux Integration Guides. He also provides Level 2 support for SAN, IBM eServer xSeries,and high availability solutions for IBM eServer xSeries and Linux. Ivo has been employed at IBM for four years.

Rob van der Heij is a VM Systems Programmer with IBM Global Services in The Netherlands. He has 20 years of experience with VM Systems Programming and has been working with Linux for S/390 since late 1999. His area of expertise focuses on VM, including running a large number of Linux images on a single S/390.

Liz Holland is a Consulting IT Specialist at the Dallas Global e-business Solution Center and is a member of the IT Specialist Certification Board. She specializes in e-business on S/390 and zSeries, supporting OS/390 WebSphere, Linux, Domino, and UNIX System Services. Her technical papers include a Redpaper on Domino performance on S/390, revisions to the VIF Users Guide, and Redbooks on Component Broker. She began working for IBM in 1980 fixing Selectric typewriters, and began supporting MVS/XA as a PSR in 1983.

Barton Robinson is president of Velocity Software, Inc. He started working with VM in 1975, specializing in performance starting in 1983. His previous publication experience includes the *VM/HPO Tuning Guide* published by IBM, and the *VM/ESA Tuning Guide* published by Velocity Software. He is the author and developer of ESAMAP and ESATCP.

Martin Söllig is an IT Specialist with Enterprise System Sales in Germany. He has 11 years of experience working in the S/390 field. He holds a degree in Mathematics from the University of Hamburg. His areas of expertise include S/390 and zSeries hardware, and major SW products on OS/390 and z/OS, with a special focus on the "new applications" on this hardware since 1999.

Simon Williams is a Senior I/T Specialist with IBM Australia. He has been working on Mainframe Systems since 1988. He specializes in System/390 and zSeries "new technologies" such as Linux, UNIX System Services and WebSphere. His technical publications include an ITSO Redbook on migrating to Domino/390 Release 5, and a Redpaper on how to install Linux for zSeries and S/390 guests under VM.

The team. Standing: Vic Cross, Michael MacIsaac, Ivo Gomilsek, Martin Soellig, Barton Robinson; kneeling: Peter Chu, Rob van der Heij, Chris Curtis, Simon Williams (Liz Holland was not available for the photograph).

Thanks to the following people for their contributions to this project:

Terry Barthel, Roy Costa, Al Schwab, Bill White
International Technical Support Organization, Poughkeepsie Center

Bob Haimowitz
International Technical Support Organization, Raleigh Center

John Eilert, Joe Temple
IBM Poughkeepsie

Alan Altmark, Romney White
IBM Endicott

Michael Kershaw
Marist University

Erich Amrehn
IBM Germany

Special notice

This publication is intended to help S/390 systems programmers and system administrators to install and manage z/VM and Linux for zSeries and S/390 systems. The information in this publication is not intended as the specification of any programming interfaces that are provided by Linux for zSeries and S/390. See the PUBLICATIONS section of the IBM Programming Announcement for Linux for zSeries and S/390 for more information about what publications are considered to be product documentation.

IBM trademarks

Comments welcome

Your comments are important to us!

We want our IBM Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  `ibm.com/redbooks`

- Send your comments in an Internet note to:

  `redbook@us.ibm.com`

- Mail your comments to the address on page ii.

Part 1

Theoretical considerations

In this part of the book, we discuss the theory behind installing and managing z/VM and Linux for zSeries and S/390 systems. For explicit examples of the work we did during this residency, see Part 2, "Practical considerations" on page 187.

1

Introduction

Then anyone who leaves behind him a written manual, and likewise anyone who receives it, in the belief that such writing will be clear and certain, must be exceedingly simple-minded.

- Plato

The introduction of Linux for the S/390 architecture has brought enterprise-level scalability and reliability to the successful Linux software platform. One area where Linux for IBM @server zSeries and S/390 has shown particular promise is in the Internet Service Provider (ISP) and Application Service Provider (ASP) market. These providers tend to run many identical servers, creating huge server farms with rack after rack of Web and application servers. In this competitive market, the cost of maintaining and supporting a large number of servers can have a substantial impact on the company's bottom line. In some geographic areas, floor space is at a premium as well. In this environment, the advantages of the S/390 are significant.

However, deploying an S/390 in an ISP/ASP setting is not a simple undertaking. Consolidating many Linux servers onto one set of hardware has unique implications for managing the individual "machines" for optimum performance.

This chapter discusses the high-level issues involved in deploying Linux for zSeries and S/390 in the ISP/ASP setting, and offers guidelines for determining the suitability of Linux for zSeries and S/390 for a particular environment.

1.1 What this redbook is

The goal of this redbook is to address some of the issues involved in managing large numbers of Linux instances on the zSeries and S/390 architecture. We include information that will be useful to ISP/ASP users who are new to zSeries and S/390, as well as to current zSeries and S/390 customers who are interested in consolidating many servers onto their existing hardware.

1.2 What this redbook is not

A growing body of other documentation already exists on Linux for zSeries and S/390, and we do not duplicate that information here; we do not cover how to install, configure and use Linux for zSeries and S/390, nor do we address specifics of the various available Linux distributions for S/390. These issues are covered in depth in the following IBM Redbooks:

- *Linux for* IBM @server *zSeries and S/390: Distributions*, SG24-6264

  http://www.ibm.com/redbooks/abstracts/sg246264.html

- *Linux for S/390*, SG24-4987

  http://www.ibm.com/redbooks/abstracts/sg244987.html

1.3 zSeries and S/390 architecture overview

The IBM @server zSeries is based on the z/Architecture, which is a new, 64-bit superset of the ESA/390 architecture. (This architecture is covered in detail in *IBM @server zSeries 900 Technical Guide*, SG24-5975.) The zSeries architecture is fundamentally different from other common systems in the ISP/ASP environment, and these differences have a profound influence on planning and implementing Linux for zSeries and S/390.

In our redbook, we highlight the ways in which the zSeries differs from Intel and traditional UNIX servers, and the impact of those differences on the Linux for zSeries and S/390 solution for ISPs and ASPs.

This section will also serve as a brief introduction to the architecture for readers new to zSeries. While it is not a complete guide to all the available options and features of the zSeries architecture, it provides a quick tour through the most important aspects that a reader familiar with existing UNIX and Intel servers would need to understand.

1.3.1 Modes of running Linux

Linux is supported on the S/390 G5 and G6 models and the z900 series. It will run G4 and older systems, but as these systems lack IEEE floating-point support, performance will be strongly affected. The following discussion briefly outlines the modes in which Linux can be run, which are illustrated in Figure 1-1 on page 5.

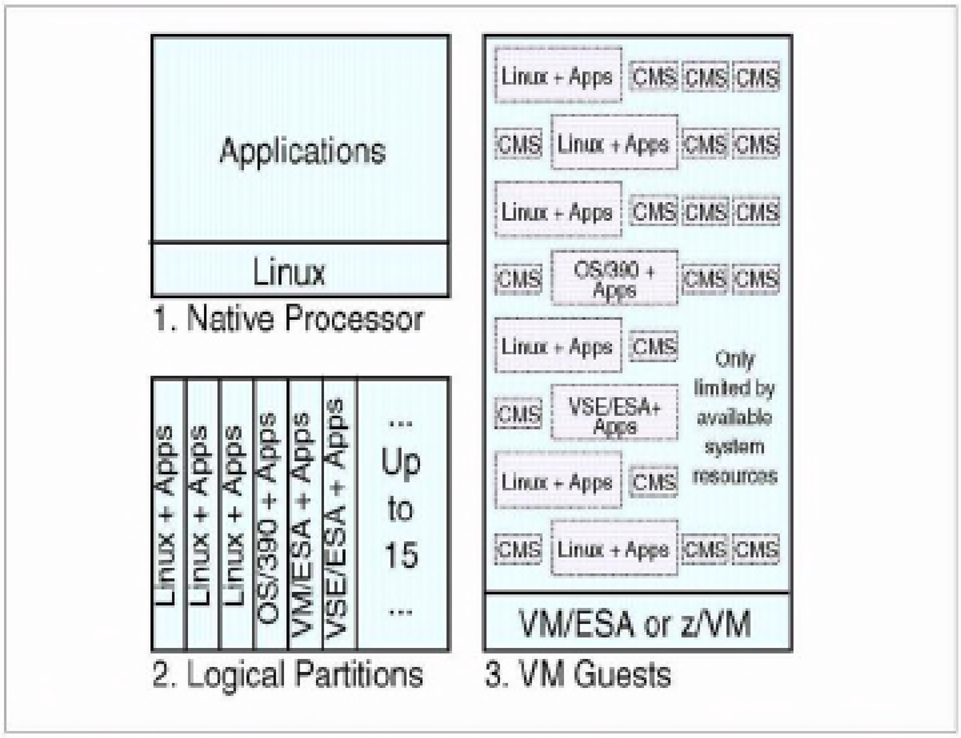


Figure 1-1 Linux for zSeries and S/390 operation modes

Native mode

The first option is to run Linux natively on a S/390 or zSeries server, as illustrated in Figure 1-2 on page 6. This indicates that only one Linux system can be run at a time, with full access to the entire system. This option has no software prerequisites other than the Linux operating system itself, and a single application that has large requirements for memory, I/O access and/or processor power can profit from the full resources of a standalone S/390 or zSeries system.

On the other hand, since there is no other operating system available for the communication between Linux and the hardware, the only possible access to the system prior to booting the Linux operating system is through a hardware console, and debugging can also be done in only this way.

Usually this choice will only be considered for testing purposes on small S/390 systems such as the P/390 or R/390. Because this solution does not exploit the abilities of the hardware and VM to share the resources of the server between several operating systems, it is of no importance for server consolidation in an ISP or ASP environment.

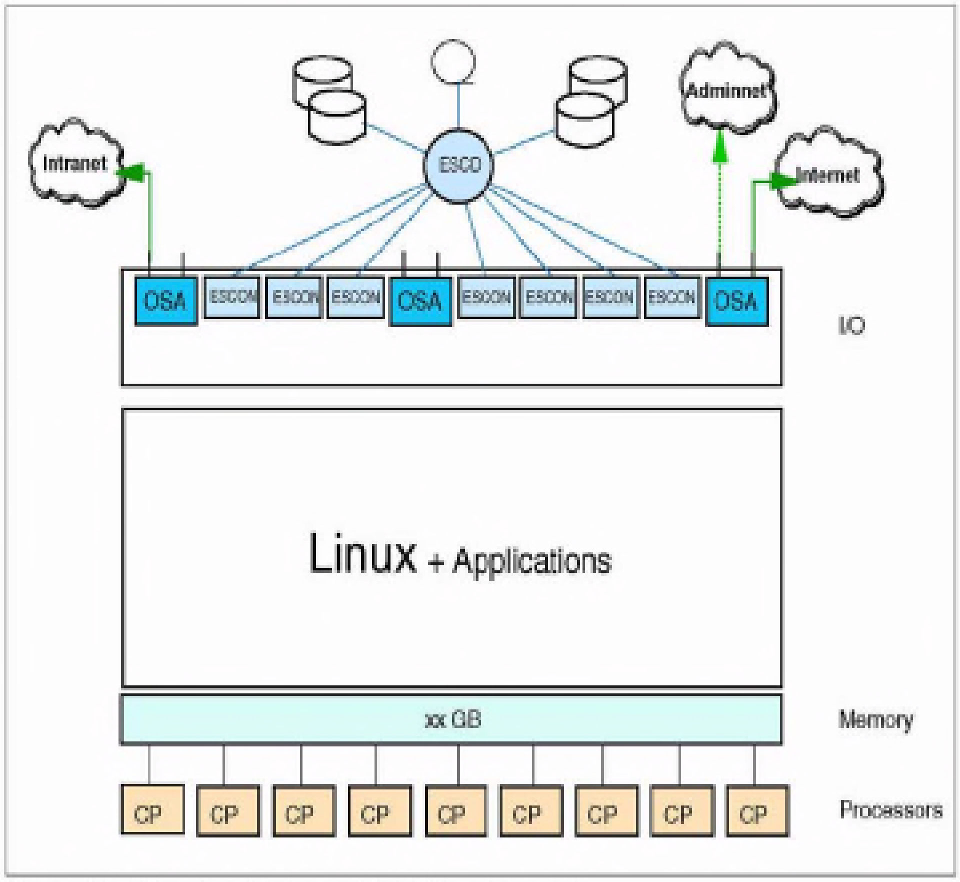


Figure 1-2 Linux for zSeries and S/390 in native mode

Running Linux in a Logical Partition

An S/390 or zSeries server can be divided into up to 15 logical partitions (LPARs), as shown in Figure 1-3 on page 7. These partitions each have their own allocation of memory, and either dedicated or shared processors, as well as dedicated and/or shared channels for I/O operations. The LPARs can operate independently of each other, since the Processor Resource/System Manager (PR/SM) microcode guarantees that operations in one LPAR are not allowed to interfere with operations in another LPAR.

Like running Linux on native hardware, no additional software is required. Additionally, the advantages of virtualization of the S/390 hardware, provided by the PR/SM microcode, can be exploited, which offers a flexibility superior to that of a static hardware solution.

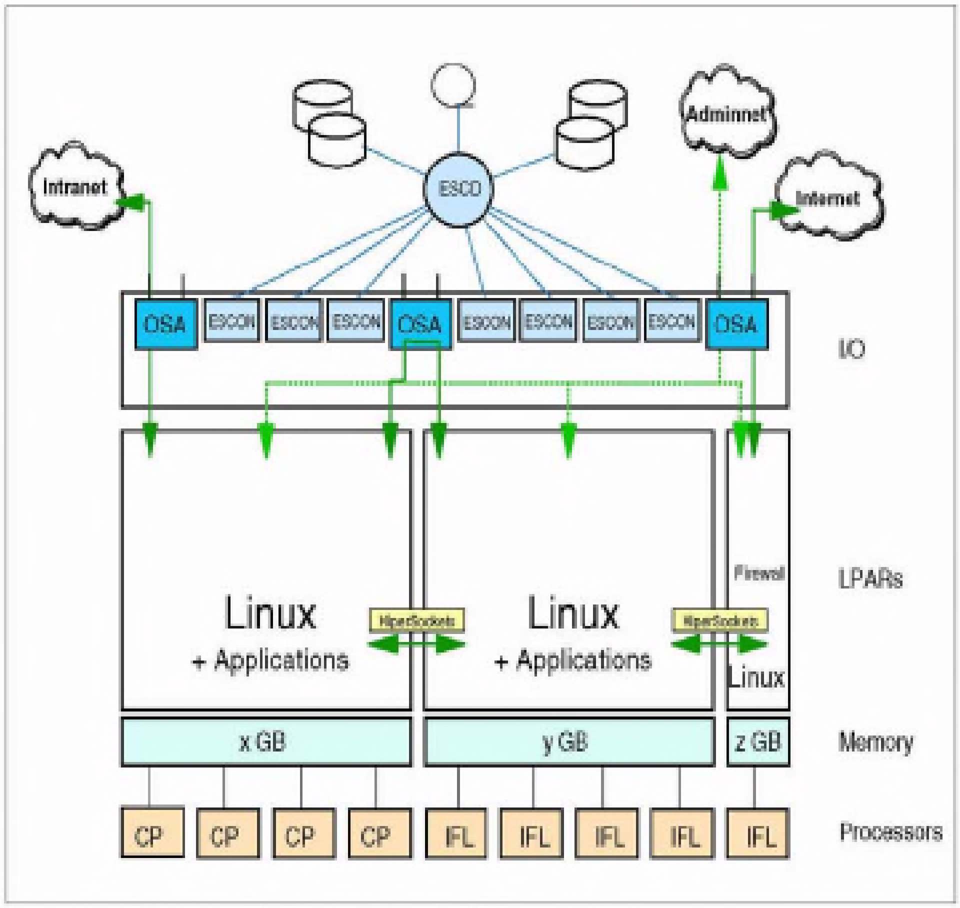


Figure 1-3 Linux for zSeries and S/390 in LPAR mode

Due to the limitation that only 15 LPARs can be defined on a S/390 or zSeries server, and that all definitions, management and debugging of these LPARs have to be done from the hardware console, this choice is also of limited suitability for server consolidation in an ISP or ASP environment. This scenario is mostly used for Linux systems for test or development purposes, when no additional maintenance of VM is wanted.

Running Linux as a VM guest

Linux can also run as a guest operating system in a VM Virtual Machine, which is illustrated in Figure 1-4 on page 8. The VM operating system provides an environment that allows you to create and manage hundreds to thousands of VM guests, in which multiple Linux images can be operated on a single S/390 or zSeries platform. Additionally a repertoire of approved VM tools and utilities is available.

Depending on the hardware platform, there are two versions of VM available: 31-bit addressing VM/ESA, which runs on 9672 or 2064 zSeries processors; or 64-bit addressing z/VM for 2064 zSeries processors, which can also run on 9672 processors in 31-bit mode.

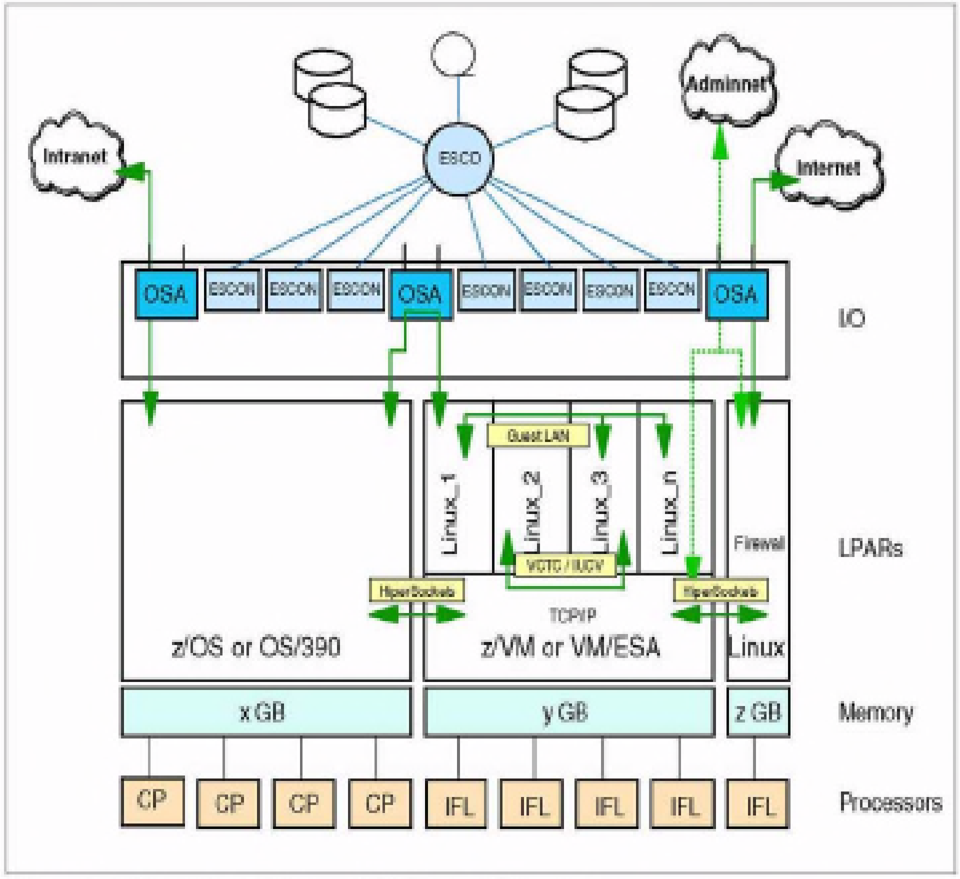


Figure 1-4 Linux for zSeries and S/390 as a VM guest

So if hundreds of virtual Linux systems have to be set up, monitored, managed, tuned and operated, using VM/ESA or z/VM is the only feasible solution. However, a certain operating knowledge of VM is required.

Normally an ISP or ASP is faced with the problem of running multiple servers with the same operating system and similar applications. These can be isolated systems or multitier applications with a database server on a central host system.

1.3.2 Processor architecture

The core of the zSeries architecture is the Multi-Chip Module (MCM). The MCM is an extremely dense package that contains either 12 or 20 Processor Units (PUs), depending on model.

Each PU can be configured to perform one of several functions, including:

- Central Processor (CP)
- Integrated Facility for Linux (IFL)
- System Assist Processor (SAP)
- Integrated Coupling Facility (ICF)

A PU assigned as a CP can be used to run any supported operating system: OS/390, z/OS, Linux, VM/ESA, z/VM, and others. By contrast, an IFL may only be used to run Linux, VIF and z/VM version 4.

SAPs are processors dedicated to performing I/O processing, offloading that task from the primary processors (CPs or IFLs), and are discussed in 1.3.4, "I/O architecture" on page 11.

The ICF is part of the Parallel Sysplex architecture, which is not currently applicable to Linux.

The associated configuration rules and options are complex, but the basic guidelines are:

- At least one PU must always be assigned as a CP.
- At least one PU must be unassigned (spare).
- 12-PU models have 2 SAPs standard, 20-PU models have 3 SAPs standard.

Thus, for a 12-PU model, there is a maximum of 9 CPs or 1 CP and 8 IFLs; for a 20-PU model, the maximum is 16 CPs or 1 CP and 15 IFLs.

Note: All processors assigned to an LPAR must be either CPs or IFLs; that is, you cannot mix them (therefore any Linux system with IFLs will have at least two LPARs - one for the minimum required CP, plus another for the IFLs).

Each PU is actually a *dual* processor, where both processors execute the same code at the same time. If the results do not match, the PU can signal that it is experiencing errors and remove itself from the active processor pool. When this occurs, one of the spare PUs will automatically be brought into service to replace the failed PU. This processor sparing is completely transparent to the application.

An important element of the MCM and PU design is the massive bandwidth available to each processor. The MCM has a total of 24 GB/sec of bandwidth, resulting in an available bandwidth to each processor of 1.5 GB/sec. This is an order of magnitude, or more, greater than traditional enterprise-class UNIX servers. This is significant because the traditional UNIX approach is to try and minimize I/O operations; using the same approach on the z900 architecture will not make maximum use of its capabilities.

Another important factor to note is that, using the MCM packaging, each z900 machine comes delivered with all 12 or 20 PUs already physically installed. The number of CPs and IFLs purchased are enabled by IBM upon installation. An advantage of this arrangement is that additional processors can be "added" by IBM as needed without taking the system down, up to the maximum allowed for that model type. (Upgrading from 9 total CPs+IFLs to 10 would require changing to the 20-PU MCM, which requires that the system be powered down.) This feature is referred to as Capacity Upgrade on Demand (CUoD).

1.3.3 Memory architecture

Note: In the zSeries and S/390 environment, the term "storage" usually refers to memory, as in central storage or main storage. In the UNIX and Intel environment, a direct access storage device (DASD), or disk drive, is considered to be storage.

It is usually clear from the context which definition is intended, but without being aware of the difference, it can be rather confusing.

Main memory

The z900 architecture supports memory configurations from 5 to 32 GB for the 12-PU models, and 10 to 64 GB for the 20-PU models. The 12-PU MCM has two memory buses and is configured with two memory cards; the 20-PU MCM has four buses and four cards.

Note: The 20-PU models have twice the memory bandwidth of the 12-PU models, regardless of how many active CPs and IFLs are configured.

The physical memory cards have 4, 8, or 16 GB of memory built in, but it is enabled in steps of 1, 2, and 4 GB, respectively. This allows a CUoD capability for memory, up to the maximum available on the installed memory cards. However, CUoD for memory is disruptive and requires a full Power-on Reset (POR).

Processor cache

z900 MCM has both L1 and L2 cache internally integrated. Each PU has 256 KB of L1 cache (128 KB instruction/128 KB data), which is similar to many other processor designs. However, the L2 cache design of the S/390 and zSeries is fundamentally different from typical UNIX server and Linux/Intel server designs. Though the L2 cache is invisible to the user and largely invisible to the operating system, the zSeries cache architecture is perhaps the most critical component of the system for you to understand in planning for, and managing, performance.

The typical UNIX or Intel server has a private L2 cache for each processor. Processors do not share L2 cache contents, and the caches themselves are fairly narrow but deep. For example, the xSeries 380 has up to 2 MB of L2 cache per processor, and each processor in the pSeries 680 has up to 16 MB of private L2 cache.

These architectures do very well when the "working set" (which is the code the processor tends to keep executing over a period of time) fits into this cache, and when the workload is easily partitioned into parallel tasks that can be assigned to a particular processor consistently. The TPC-C benchmark is an example of a workload that is well-suited to this type of architecture.

However, in general these servers have relatively limited memory bandwidth, so that the more frequently cache misses occur and data must be retrieved from main memory, the less the deep, private cache helps. In particular, when the system is heavily loaded and tasks must compete for processor time, each task's working set must be loaded into the private cache each time that task moves to a different processor. It is for this reason that most SMP UNIX servers are typically sized to run at utilization levels of approximately 40 to 50%.

By contrast, the zSeries has two 16 MB L2 caches which are shared by 6 and 10 PUs each (in the 12-PU and 20-PU models, respectively). These caches are also highly connected to each other and contain a great deal of interconnect logic, in addition to the cache memory itself. With this shared cache, tasks can be moved from processor to processor as needed to manage load without necessarily incurring a cache miss.

In addition, when cache misses do occur, the significantly larger memory bandwidth available minimizes the cost of retrieving data from main memory. Thus the zSeries cache design tends to favor applications that do not have cache-friendly working sets and behavior. The shared cache design also explains why zSeries servers are typically run at much higher utilization than UNIX servers; loads of 80 or 90% are not uncommon.

The MCM, memory cards, Cryptographic Element (CE) chips, Distributed Converter Assemblies (DCAs) (processor cage power supplies), and External Time Reference/Oscillator (ETR/OSC) cards are installed in the Central Processor Complex (CPC) cage.

1.3.4 I/O architecture

Because the options and configurations available for I/O on the z900 are quite complex, a comprehensive description of them is beyond the scope of this redbook. However, we touch upon the subject briefly in this section, since I/O on the z900 is fundamentally different from typical UNIX or Intel server I/O.

Channels and subchannels

The basic building block of I/O connectivity on the z900 is the *channel*. Conceptually a channel can be viewed as a path from the processor (MCM) to a physical device. Indeed, for external devices (DASD, for example) a channel eventually translates to a physical cable or fiber.

A *subchannel*, by comparison, can be viewed as a logical connection from a specific process to some I/O resource. For example, a specific network interface in a process is connected to a subchannel. Many subchannels are carried over a single real channel.

System Assist Processors (SAPs)

SAPs are PUs configured to run specialized microcode, and they manage I/O operations on behalf of a CP or IFL. The SAP selects the appropriate channel to use for the I/O operation, and is also responsible for scheduling and balancing to optimize access to devices.

From the CP's perspective, the I/O request is handed off to the SAP, and the SAP informs the CP when the data transfer to or from main storage has been performed and the I/O operation is complete. The entire process is transparent to the programs running on the CP (e.g. the Linux guests), but it means that the CP is freed from dealing with the details of I/O.

Self-Timed Interfaces (STIs)

The zSeries and S/390 servers do not have a traditional bus architecture like Intel servers and many UNIX servers. Rather, the MCM has 24 Self-Timed Interface (STI) ports. Each STI on the zSeries has a maximum bandwidth of 1 GB/sec full-duplex. An STI port can be connected via copper cable to an STI-Multiplexer (STI-M) card in the z900 I/O cage. Each STI-M card creates an I/O domain of four slots, with one secondary 333 MB/sec STI link per slot.[1] Each slot can hold one of the following I/O cards:

- ESCON-16 channel cards
- FICON channel cards
- OSA-Express (OSA-E) channel cards
  - Gigabit Ethernet (GbE)
  - Asynchronous Transfer Mode (ATM)
  - Fast Ethernet (FENET)
- PCI-Cryptographic Coprocessor (PCI-CC) cards

[1] The secondary STI link operates at 500 MB/sec for the InterSystem Channel-3 (ISC-3); this is used for Parallel Sysplex operations and is not applicable to Linux.

Note that there is also a Compatibility I/O Cage available, which uses cards designed for the G5/G6 series of servers. In this case the Compatibility I/O Cage is connected via an STI-H multiplexer card plugged into the CPC, providing 333 MB/sec ports. The Compatibility I/O Cage is necessary to use any of the following cards:

- OSA-2
  - Token Ring
  - FDDI
- ESCON-4
- Parallel-4
- Parallel-3

ESCON and FICON

The Enterprise Systems CONnection (ESCON) channel is the current standard interface for connecting external I/O devices to zSeries and S/390. It is a 20 MB/sec half-duplex serial bit transmission interface carried over fiber optic cables.

ESCON ports are provided by ESCON-16 cards, which provide 15 active ports plus one spare per card. The ESCON-16 cards are always installed in pairs, and ports are activated in ordered blocks of four ports (e.g. if four ESCON ports are ordered, then two ESCON-16 cards will be installed, each with two ports enabled).

The FIber CONnection (FICON) was introduced on the G5/G6 S/390 and zSeries servers to provide higher bandwidth and increased connectivity. FICON provides a 100 MB/sec, full-duplex serial interface over fiber optic cables. In addition, FICON allows multiple I/O operations to be outstanding at the same time to different channel control units. Taken together, these new features allow one FICON channel to provide the same I/O concurrency as up to eight ESCON channels. FICON cards provide two FICON ports each.

1.3.5 Network architecture

zSeries and S/390 network connectivity is provided by the Open Systems Adapter-2 (OSA-2) and OSA-Express (OSA-E) interfaces. These interfaces provide full TCP/IP connectivity to a broad selection of industry-standard networks. Several OSA-2 cards are supported by the zSeries; the newer network interfaces are supported by the OSA-E cards.

OSA-2

OSA-2 cards are the previous generation of network interfaces, and are supported as standard channel-attached devices.

The OSA-2 family consists of the following cards:

- OSA-2 ENTR (Ethernet/Token Ring)[2]
- OSA-2 FENET (Fast Ethernet) - not supported on zSeries
- OSA-2 FDDI (Fiber Distributed Data Interface)
- OSA-2 ATM (Asynchronous Transfer Mode) - not supported on zSeries

For the Ethernet, Fast Ethernet, and ATM interfaces, there are new OSA-Express features available that support these network types, in addition to some new types.

OSA-Express

The OSA-Express cards are the new generation of network interface and are supported on both the G5/G6 series S/390 (one port per card) and the zSeries (two ports per card).

The OSA-Express features introduce a new operating mode, Queued Direct I/O (QDIO). QDIO is a highly efficient data transfer mechanism that significantly increases data throughput. It uses shared memory queues and an internal signaling protocol to exchange data directly with the TCP/IP stack. The OSA-Express features appear as channel types OSE for non-QDIO mode, and OSD for QDIO mode. The following types of OSA-Express features are supported:

- OSA-Express GbE (Gigabit Ethernet)
- OSA-Express FENET
- OSA-Express ATM[3]

1.3.6 Disk architecture

The zSeries and S/390 have many different options for connecting fixed-disk storage (DASD). In general, they all appear as channel-attached devices using ESCON or FICON ports, though some older systems may use the IBM bus and tag parallel interface.

The current recommended storage device for the zSeries is the IBM Enterprise Storage Server (ESS), also known as Shark.

[2] The zSeries only supports Token Ring mode for OSA-2 ENTR.

[3] It only supports QDIO mode in ATM LAN Emulation (LANE) mode.

ESS/Shark

The IBM Enterprise Storage Server (ESS) is a complete disk storage system that provides up to 13.9 TB of managed storage to all major types of servers. The ESS uses large caches and two 4-way SMP RISC processors to provide extremely high performance. It can be integrated into Storage Area Network (SAN) architecture. Multiple servers can be attached to the ESS using the following types of interfaces:

- Fibre Channel
- ESCON
- FICON
- UltraSCSI

More information about the ESS is available at:

http://www.storage.ibm.com/hardsoft/products/ess/ess.htm

1.3.7 Models

The z900 family of systems is designated by a model number of the form 2064-101. The last two digits of the part number designate how many central processors (CPs) are enabled on the machine; thus the 2064-101 has one CP enabled; the 2064-116 (the largest model) has 16 CPs enabled. There are also capacity models which have part numbers of the form 2064-1C1. These models are intended to be ordered to support Capacity Backup, and are 20-PU models. The -1C6 model, for example, has 6 CPs enabled.

1.4 Solution applicability

One of the side effects of consolidating many discrete server workloads onto one zSeries server is that the applications interact and affect each other in ways that they do not in a discrete environment. Accordingly, some workloads types are more appropriate matches and are likely to be "good citizens" of our penguin colony.

1.4.1 Better matches

Based on the characteristics of the z900 architecture previously described, workloads that are excellent candidates for migration to Linux for zSeries and S/390 are ones that exhibit some of the following characteristics:

- I/O-intensive operations (e.g. serving Web pages)
- Lightly-loaded servers
- Custom-tailored system images (that is, no "default installs")

1.4.2 More difficult matches

By the same token, some types of workloads are not well-behaved in the Linux for zSeries and S/390 environment and thus are poor candidates:

- Compute-intensive operations (technical computing)
- Graphics (X-Window system, etc.)
- Heavily-loaded servers
- Tasks that check the system clock often (to see if configuration files have changed, for example)

Floating-point calculations can be especially problematic for older machines; prior to G5, there was no support for the IEEE floating-point format in the ESA/390 family. As all Linux floating-point expects IEEE, these machines are especially poor performers for graphics and other computational loads. The G5 family introduced IEEE support via microcode; the G6 family received a significant floating-point performance boost by moving IEEE support into actual silicon.

1.5 z/VM and why you want it

Linux for zSeries and S/390 can be run in several different modes, each with its own advantages and disadvantages. At the present time the available modes are Native, LPAR, and z/VM[4]. These options are discussed briefly in the IBM Redbook *Linux for S/390*, SG24-4987.

We strongly recommend that z/VM be used to deploy Linux in the ISP/ASP environment. There are some additional complexities and skills that will need to be learned, but we believe that z/VM offers an extremely powerful environment which is necessary to successfully manage hundreds or thousands of virtual Linux servers. Some of the features z/VM brings are:

- Resources can be shared among multiple Linux images, including CPU, memory, storage, and network adapters.
- New guests can be added quickly, without requiring dedicated resources.
- There are extremely high-speed virtual networks between guests.
- It provides centralized storage management and backup.
- A rich debug and test environment allows images to be created that duplicate production systems, without requiring additional physical resources.
- It provides comprehensive workload monitoring and control facilities.

[4] With the release of z/VM 4, we no longer recommend the Virtual Image Facility (VIF) as a configuration option. z/VM version 4 removes many of the limitations present in VIF and has additional functionality specifically designed to support Linux for zSeries and S/390.

1.6 Skills and resources required

While running Linux for zSeries and S/390 has many advantages in terms of potential reductions in staffing requirements and hardware management costs, this does not mean that it is a simple undertaking. Running hundreds or thousands of servers is a complex undertaking in any environment and requires qualified, knowledgeable staff to implement successfully.

Each phase of the deployment of Linux for zSeries and S/390 in the ISP/ASP requires a slightly different mix of skills. To a certain extent, the requirements will vary from company to company, depending on existing business processes and systems in place. The general guidelines offered here may offer some direction.

1.6.1 Planning and installation

This is by far the most critical phase of any ISP/ASP system deployment. The decisions taken during this phase can have a dramatic impact on the future performance and ultimate scalability of the system. This is especially the case for Linux for zSeries and S/390, where the system itself serves as infrastructure for many virtual servers. The infrastructure must be solidly in place and well-designed in order for the guest servers to also be stable and efficient.

For this phase the following individuals should be heavily involved (some roles may be filled by the same person):

- Linux system administrator
- Network administrator
- z/VM system programmer
- Technical support manager
- Sales and marketing representative

The Linux system administrator and network administrator fulfill much the same roles they would in any normal multiserver deployment: assigning IP addresses, designing subnets, determining disk layout, and so on.

We believe that the addition of a technical support manager and sales/marketing representative to the planning team is important to the ultimate success of the project. Both technical support and sales departments need to be aware of characteristic advantages and limitations of the virtual server environment to properly inform and support their customers.

One of the most crucial members of this team is the z/VM system programmer. Unfortunately, this is also likely to be a difficult position to staff for most ISP/ASP operations. There are many parts of the system that need to be configured appropriately, and this requires a relatively high skill level with z/VM. Each system is slightly different, therefore generic "cookbook" approaches are less likely to be successful. It may be possible to acquire this expertise on a temporary contract or consulting basis during the planning phase, and brought in as needed once production operations begin.

1.6.2 Linux image deployment

Once the system is installed and configured, the deployment of new virtual servers is much less complicated. The network administrator will still be responsible for managing IP addresses and network architecture, and the Linux system administrator is responsible for configuring each individual server. However, the Linux system administrator needs to develop some z/VM skills to maximize the efficiencies of the system. Ideally there will be a fair degree of automation in place to create new images, as we detail later in this book.

1.6.3 Maintenance

Maintenance will typically be the domain of the normal operations support staff. In most cases we anticipate that typical UNIX server monitoring and maintenance skills are readily transferable to the zSeries environment, with minimal additional training in some specifics of the z/VM monitoring and control functions.

2

Sizing

As effective as zSeries and z/VM are in their ability to run large numbers of Linux guests, there are of course limits to how much work a given system can handle. Accordingly, in this chapter we discuss guidelines concerning sizing for your applications and workloads.

This includes topics such as CPs, storage, DASD, and network bandwidth, as well as a general overview of the reasoning behind some accepted sizing methodologies and our assessment of their validity.

2.1 The nature of sizing

First of all, it's important to realize that despite the best efforts of many experts both within and outside of IBM, sizing is an inexact science, at best. A sizing estimate is an *approximation* of the hardware resources required to support a given workload. This estimate is drawn from the best information available at a point in time, in order to obtain a base to work from.

But of course, the actual result will often differ from the results of the estimate, due to factors such as imperfect data (the Garbage-In, Garbage-Out scenario), workload skew not being accounted for in the sizing, custom code, and other customizations.

Add to this the fact that lab conditions do not fully correspond to real world production environments, and it becomes obvious why sizing results cannot be guaranteed. This is not to say sizing cannot be done, of course, but rather to emphasize the fact that there is always a certain margin of error involved.

This is particularly noteworthy in our case because of the differences in the performance characteristics between Linux on the x86 and Linux on the zSeries platforms. As noted in Chapter 1, "Introduction" on page 3, the processor, memory, and I/O architectures of the zSeries platform are substantially dissimilar to the platforms Linux has historically been run on.

Therefore, extra care is needed when performing sizing estimates for running Linux on the zSeries, as the shift in relative capacity of the systems may be larger than expected. Also, one should keep in mind that sizing is really just a "first guess" based on the best available data, and should always be followed up by further analysis.

This also means that you need to understand relative capacity across server architectures, which serves as the basis for sizing. This subject is covered in the next section.

2.1.1 Sizing vs. capacity planning

Before we move on, we need to provide a clarification: some readers may be familiar with the term *capacity planning*. This is a service that IBM has been offering for many years to its customers. On the S/390 platform, this involves getting SMF data (and projected growth rates) from the customer's installation, and then modeling the combination to provide an understanding of how to handle the growth.

However, this is *not* what we mean by sizing.

Capacity planning is relatively predictable because it involves migrating well-understood workloads with measured usage data within the same architecture. However, when we size Linux applications for the zSeries, not only are we moving across very different platforms, but are often doing so with little or no application data available for analysis to support the sizing.

Given these limitations, readers who are used to the methodology and results of capacity planning should reevaluate their expectations for system sizing. To reiterate: sizing is definitely an area where the margin of error tends to be on the large side. But enough about limitations; let's see what sizing can do for you.

2.2 Relative system capacity

To really understand relative capacity, we need to understand the balance of processor power, the internal bandwidth per processor, and the relative ability of the operating system to schedule work in a way that efficiently utilizes resources.

When estimating the relative capacity of servers, most people base their calculations on the correlation between processor speed and system capacity.

For example, many home PC users tend to regard the clock rate as the prime indicator, but this is too simplistic, not to mention misleading, due to microprocessor design differences (witness the recent battle between Intel and AMD, or the older x86 vs. PowerPC conflict, for some high-profile examples).

Other users turn to benchmarks. Some use a processor-oriented benchmark like SPECint, but differences in the memory and I/O subsystems are masked. Some use an industry standard or published application benchmark like TPC-C or SAP R3 SD, but the presence of workload imbalances (called "skew") and variance in the amount of data handled ("working set size") make these imperfect measures, as well. We examine benchmarks in more detail in the next section.

Attention: In the mainframe environment, people often used the term MIPS to indicate relative capacity. MIPS is an acronym for "Millions of Instructions Per Second", but it has also been called a "Misleading Indicator of Processor Speed", and even "Meaningless Indicator of Processor Speed".

There is perhaps some justification for these terms because instruction rate has only a casual relationship to MIPS ratings, which are actually relative throughput metrics on a scale where the S/370 158 was considered 1 MIPS. Mainframes have been designed, with a balance of internal bandwidth, I/O performance, processor speed, and scheduling capability, to achieve their MIPS ratings on the workloads called Large Systems Performance Ratios (LSPRs), which have become well understood over time. LSPR ratios are *not* simply a measure of relative processor speed, even though they are often assigned a metric called MIPS.

The term MIPS has caused many people to compare instruction rates of various processor architectures as a way to compare server capacity. But because mainframe MIPS ratings are not really instruction rates, they cannot be used directly in this manner.

2.2.1 Benchmarks

Traditionally, benchmarks have played an important role in migration and workload planning. If you're going to deploy new applications or port existing applications to a new platform, you'll want to know how the new system will perform—this where benchmarking comes in.

Well-established organizations such as the Transaction Processing Performance Council (TPC) and the Standard Performance Evaluation Corporation (SPEC) have long maintained sets of performance metrics with which to evaluate the many platforms that are commercially available: These organizations are on the Web at:

http://www.tpc.org
http://www.spec.org

However, these benchmarks are not very useful, for two basic reasons:

- At the time of writing, IBM has not released any benchmarks for Linux on the zSeries.
- Benchmarks by nature tend to be artificial, and in some cases misleading.

The second comment may be considered a bit controversial, so let's discuss further what we mean by "artificial" and "misleading".

Real throughput by a system is the result of many factors. Processor design and clock speed (collectively referred to here as processor power), internal bandwidth for memory and I/O, and scheduling of work in order to effectively utilize available resources (done by the operating system) are the primary considerations. Therefore, system capacity can only be effectively compared when all these factors are taken into account.

Figure 2-1 shows a graphical representation of this concept.

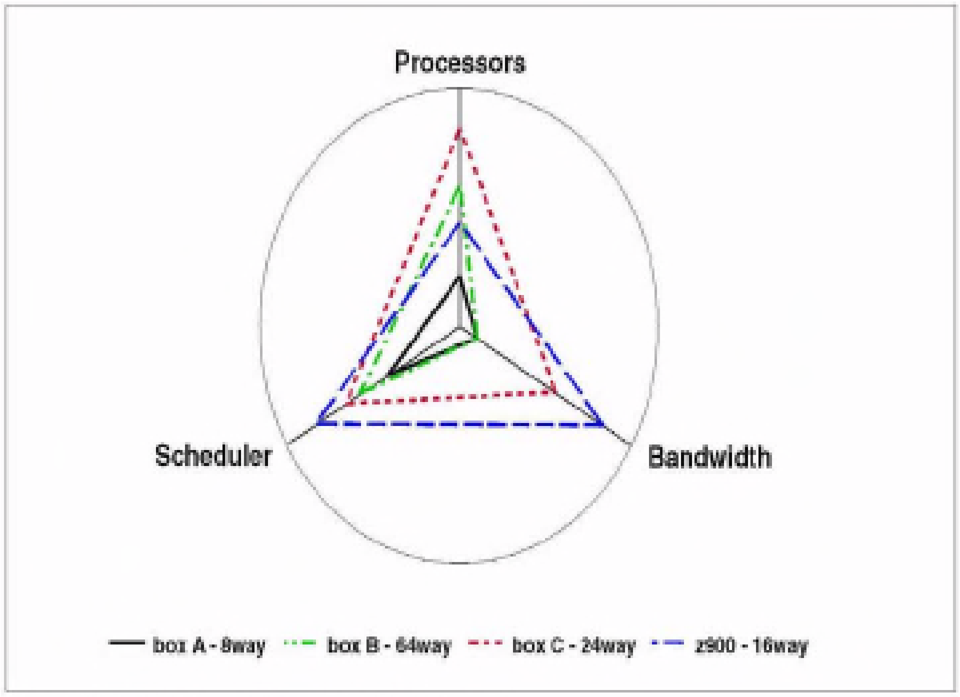


Figure 2-1 Relative system capacity on 3 axes

The misleading part comes into play when we realize that some benchmarks do not look at the "big picture" of the system. For instance, the SPECcpu benchmark, true to its name, focuses almost exclusively on the processor. In other words, it is only looking at the vertical axis of this graph, which represents the processor power of the engines in question, multiplied by the number of engines in the system, as noted in the legend.

But with this approach, bandwidth (which measures the data rate per processor between the cache and memory), and scheduling (which represents processor utilization typical to the particular system), are both ignored. What good is a blazing fast CPU that is sitting idle? The slowest processor in the world can do nothing just as quickly as the fastest.

So if processors, bandwidth, and scheduler are equally stressed by the workload, the relative capacity of the machines can be represented by the geometric mean of the parameters. This is shown in Figure 2-2 on page 25.

If the workload shifts *away* from balance, then relative capacity also shifts. Thus, for CPU-intense environments where the parallel portion of the work is in balance (also known as "skewless"), the relative scaling will shift away from the mean toward the machines with more and faster CPUs. When significant data stress (large "working set", mixed workload, large user counts, etc.) or skew (data sharing, variable usage patterns, "spikiness" causing workload imbalances) is present in the workload, the relative scaling will shift in favor of machines with higher internal bandwidth per engine and better scheduling.

The result is that the relative capacity of machines will vary significantly from workload to workload. Figure 2-2 on page 25 illustrates the difference between the relative capacity indicated by most commercial benchmarks (the processor bars) and a workload which has enough data stress and/or skew to move the capacity toward the geometric mean of processor, internal bandwidth, and scheduler, using the data from Figure 2-1 on page 23.

You can see that the IBM machines are more balanced than the others, and also that the S/390 does not fare well with CPU-intense work. This follows intuitively from the conflicting views offered by the proponents and opponents of the machine: scaling on the processor bars indicates that the machine is "big", while scaling by the bandwidth bars indicates that it is "slow".

The situation is compounded by the fact that most people work with workstations or PCs, where processor speed has a much higher leverage on total capacity to do work. But because servers need to respond to many users, there is much more "context switching" in their workloads than in that of a PC. This means the internal bandwidth and scheduling (the second and third bars) become more critical to the capacity to do work, and to do it quickly.

Understood in this sense, then, system capacity is measured by the size (or, geometrically speaking, the area) of the triangles seen in Figure 2-1 on page 23. No longer are you restricted by a single aspect of the system's performance.

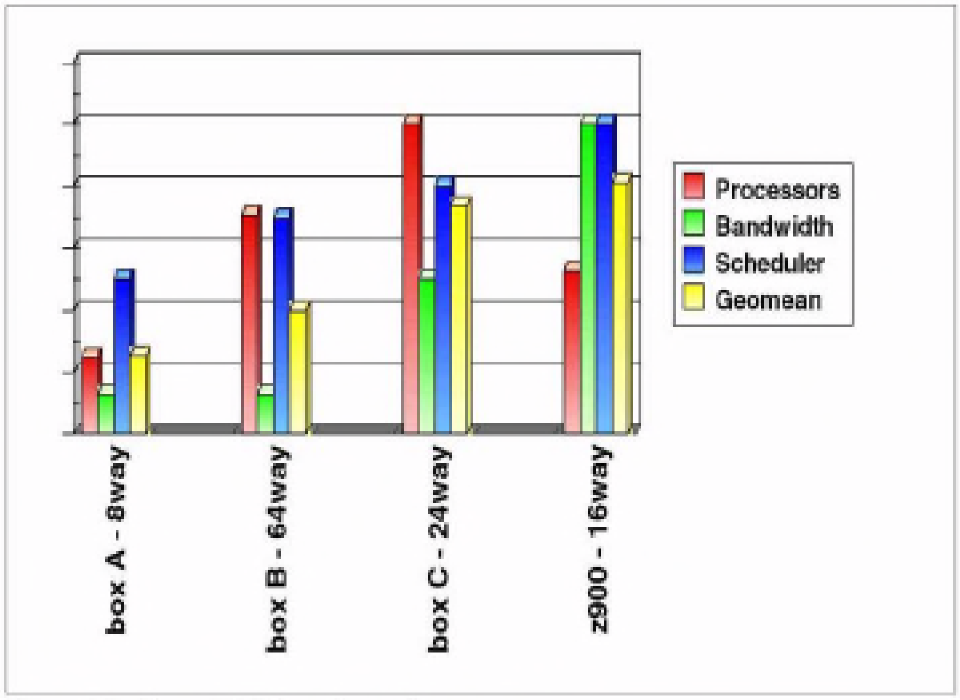


Figure 2-2 Geometric Mean of sample systems

2.2.2 The bottom line

The bottom line of this discussion is that the relative capacity of machines varies with workload. This is most dramatic when it comes to the S/390, but differences also show up between "enterprise" and "midrange" UNIX servers, UNIX and NT servers, older and newer servers, etc.

It is true that within a machine type (family), relative capacity is close to the relative processor speed, unless a bottleneck develops. But this is not true for systems with different architectures. Therefore, benchmarks which are reliable indicators in comparing like machines cannot be used to compare different machine types with the same confidence. Since the objective is to make the best possible decision when choosing a server for a given workload, it is important to consider how the characteristics of the workload match the design structure and capabilities of the various machines.

For these reasons, it is impossible to simply position the various servers in a list from largest to smallest. IBM's large commercial servers have more robust scheduling and higher internal bandwidth per processor than other servers, allowing them to maintain high processing rates in the face of skew. As a result, they will have higher capacity than their competition on many real workloads.

To address the issue of workload dependence, IBM has put resources in place to provide sizing, capacity analysis, and capacity planning assistance. If you need a sizing performed, contact Techline at (888) 426 5525, or fill out a TechXpress form on the following Web site:

http://dalnotes1.s1.dfw.ibm.com/atss/techxpress.nsf/request

2.3 Utilization

Linux on zSeries is really about doing the work of many smaller servers on one (or a few) larger servers. This is typically done to gain advantages in total cost. In most scenarios, the advantage is realized by reductions in the growth of floorspace, power, people, network complexity, etc. It turns out that server consolidation is most viable when there is some inefficiency in the current operation. This can come about in a variety of ways, and no two total cost scenarios will be exactly the same. However, returning to the capacity effects of server consolidation, we observe the following.

Clearly, we are not going to be able to show consolidation viability if the application is CPU-intensive enough to show the S/390 engines at a disadvantage. However, this does not turn out to be the biggest lever. Utilization can have up to a tenfold (or more) leverage on relative capacity. This is because distributed servers are designed for the individual peaks of the various workloads in question. This level of capacity is an overdesign for the composite peak of the workload, unless all workloads peak simultaneously, which almost never happens.

Furthermore, as we have seen, response time on some servers and workloads can be very sensitive to utilization. When servers are inexpensive, there is a tendency to buy more when the response time starts to grow, regardless of the actual load on the machine. In some cases, this can happen at 40% or 50% utilization, resulting in many machines with very low average utilization.

Then there are all the other servers which tend to go with a production system - backups, test, development, and infrastructure servers such as file, print, DNS, security, system managers, and so on.

When all these are added together in a composite utilization picture, the utilization can be quite low, even at the peaks.

2.3.1 White space - unused capacity

Since utilization is a major factor in assessing the effect of consolidation, we need to understand it better. We also need to know its inverse, which is the "white space", or unused capacity of a system.

We start with definitions. This is particularly important because utilization is defined by how we measure it. All machines contain a counter that is incremented every cycle. This counter is used to generate a running notion of time within the system. The hardware counter is not infinitely long, but it is typically rolled over into a software counter kept in storage that is sufficiently long that we don't have to worry about it rolling over. Most machines have another counter which increments only on cycles when the processor is busy (this includes when it is busy waiting for memory on a cache miss). This counter is also extended by software and the count of busy cycles is kept in memory. Utilization is defined as the change in the "busy count" divided by the change in the total "cycle count".

Since cycles occur at a fixed rate in time, the change in cycle count is a measure of a time interval. Thus, the software that generates utilization data operates by periodically checking the change in the busy count. The cycle count in the denominator is determined by how often the software is run. This in turn is controlled by starting the data gathering code at regular intervals by dispatching it when the change in the cycle counter indicates that an interval has completed. When it runs, the data gatherer reads both counters, does the division, and stores or displays the result for use to look at.

White space is simply defined as *1 - utilization*. Thus, utilization is a statistic which is always an average or probability. This is why we need to be careful when we talk about peak and average utilization. The shorter the interval used to gather the utilization data, the more the utilization number looks like either one or zero. This is because if we get to the ultimately short interval of one cycle, the machine is either busy or it is not. Moreover, the shorter the interval used to gather utilization data, the more impact the gathering of data has on what we are measuring. In the most extreme case, the software that does the gathering keeps the rest of the system from running and the utilization becomes 100%, even though the throughput has gone to zero.

Because of these factors, utilization graphs will look spikier when the interval is short, and smoother when it is long. Because it is easier to see the white space on smoother graphs, and because the impact of gathering statistics on longer intervals has lower impact on the system, we typically look at 1- to 15-minute intervals when gathering statistics.

Is this enough? See the utilization profiles in Figure 2-3 on page 28, which all have 50% white space.

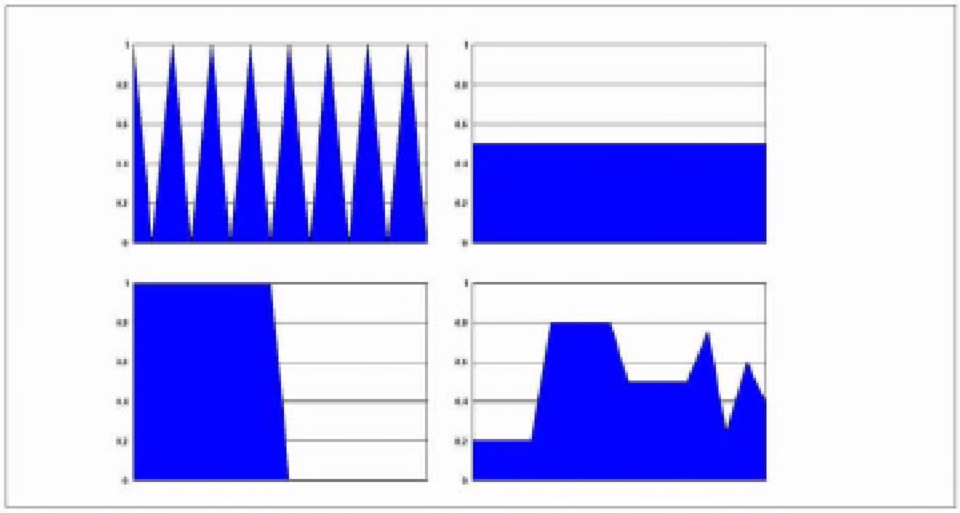


Figure 2-3 Sample utilization charts with 50% white space

Obviously, the average utilization is not enough, because the peak of each workload is also of interest. One one hand, we need to have enough capacity to handle the peak, but we'd also like to minimize the white space in order to run more efficiently. In each of these cases, 50% of the capacity goes unused.

When we build a composite of all 4 workloads, we get the graph shown in Figure 2-4:

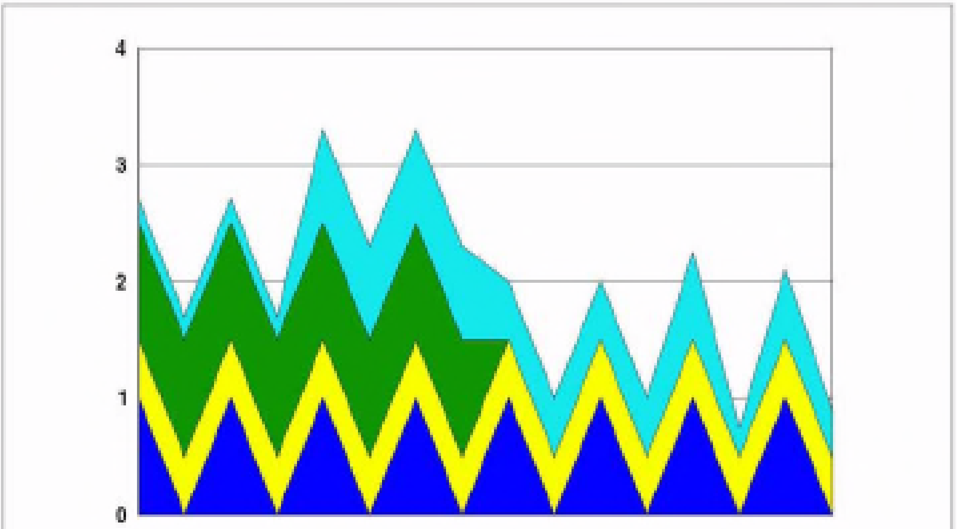


Figure 2-4 Utilization after consolidation - also 50% white space

Here we see that the peak workload is about 80% of the total individual configured capacity. If this peak holds up, we probably are not getting a large lever from consolidating. However, often such peaks occur because of scheduled batch windows for data loads, reports, backups, etc, which can be prioritized or rescheduled using white space in different periods, thus smoothing the curve and reducing the peak.

The example of 30+ servers shown in Figure 2-5 illustrates this case; the peaks are identified as database backups that are all scheduled at the same time on the distributed solution, but can be staggered to reduce the peaks on the consolidated machine. In this example the distributed solution has about 62% white space, meaning that over half of the configuration goes unused.

Some of this occurs naturally, even if the individual systems are efficiently configured, because individual systems must be configured for each workload's peak; this is shown as the grey space on the chart. Only 30% of the distributed configuration is "headroom", even though 60% of the composite is white space. This means that even if 30% headroom is maintained, the composite can be built with 30% less total capacity.

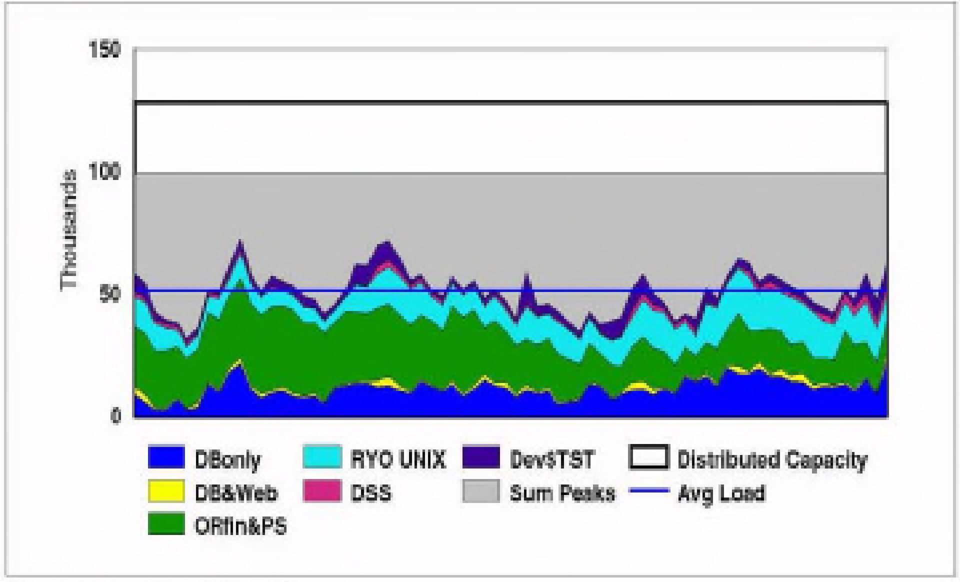


Figure 2-5 Using Virtual Servers

In this particular case, the composite utilization is actually quite high for a distributed solution. In many other cases, there is even more white space. For example, Figure 2-6 on page 30 shows the composite of 147 servers in IBM's "universal server farm" run by IGS as part of their Web-hosting operation. Here, the composite peaks at 13%—this means there's over 87% white space in the configuration.

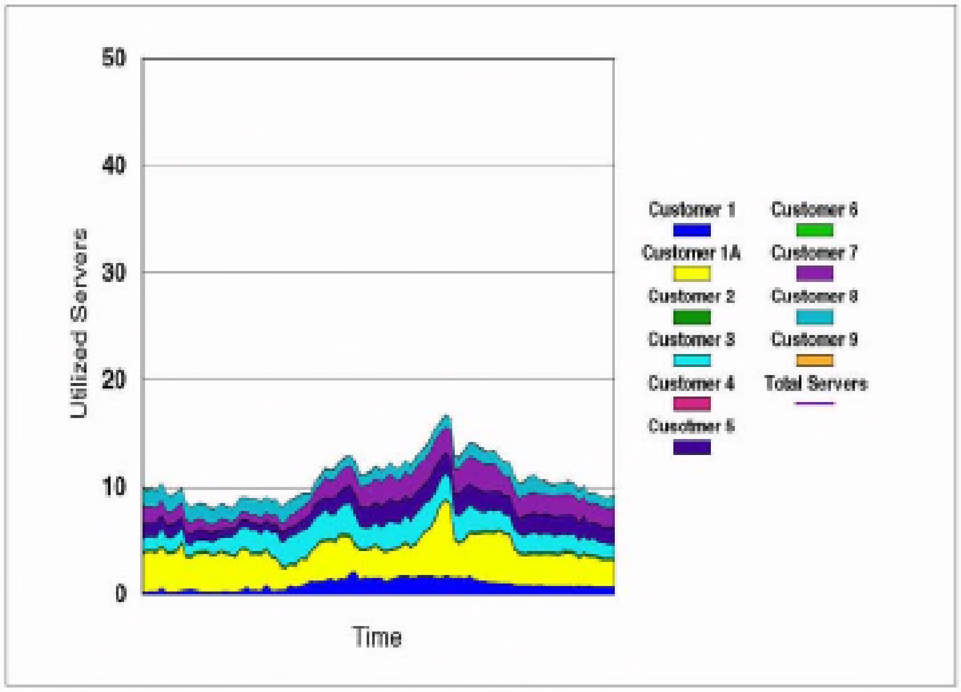


Figure 2-6 Web servers - consolidation candidate

Sources of white space

So where do such large inefficiencies in the deployment of computer power come from? White space comes from six main sources:

- Spikes
- Headroom
- Redundancy
- Fragmentation
- Partitioning
- Skew

Spikes

"Spikes" in a workload result from the variance of its demand for service over time. If the demand is very variable and the work is high priority, it is difficult to use the white space, because the work that fills the troughs must be overridden by the high priority spike.

This has several implications. First, when white space is caused by spikiness, there must be a scheduler in place that can dispatch work according to priority and can also guarantee that dispatched low priority work cannot inappropriately hold the resources. Second, the presence of spikiness indicates that the context switching capability of the machine becomes important. Figure 2-7 on page 31 shows the relative context switching capability of Linux for S/390 and Linux for Intel.

Rapid context switching of large context changes is a characteristic of workload mixing that occurs when work is added to a system to use white space. The S/390 hardware design is particularly well-suited to this environment. It is the underlying shared L2 cache and high memory bandwidth per processor which enables this, and by extension, the ability to run virtual machines.

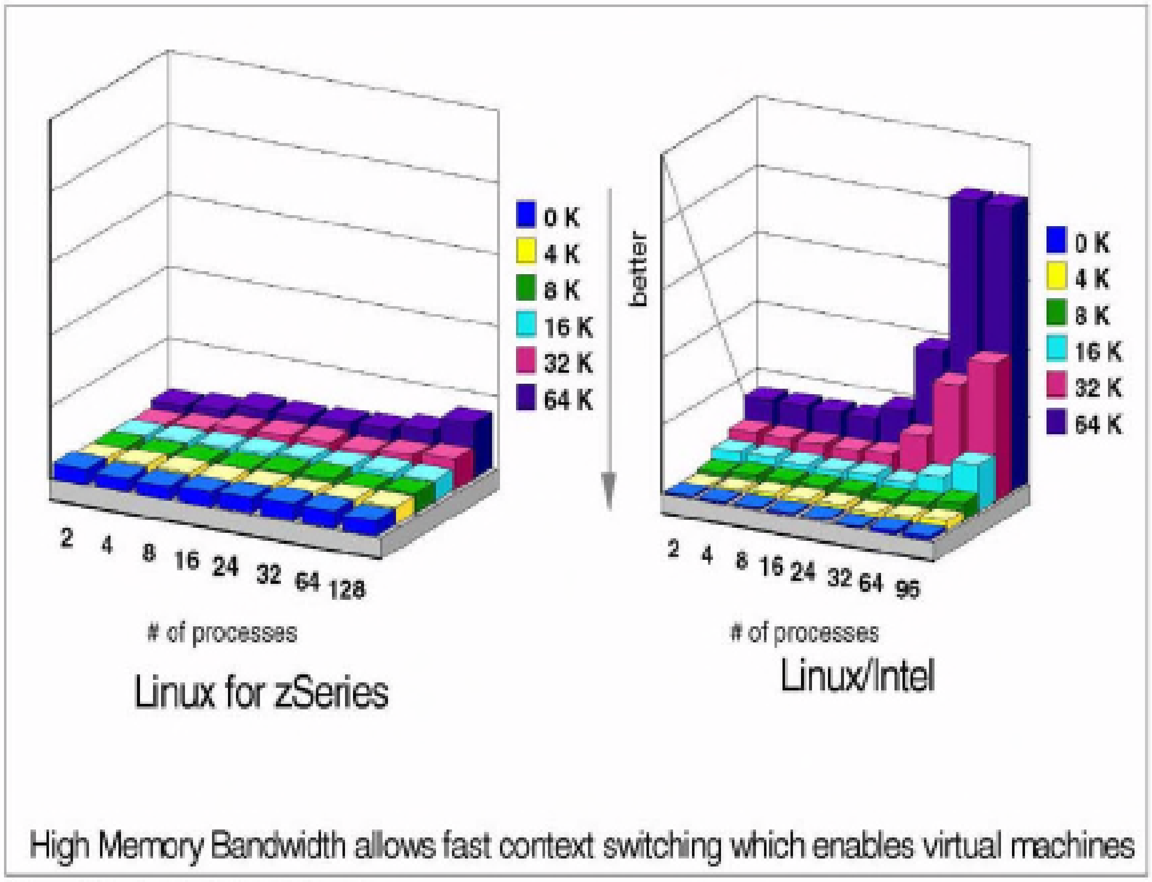


Figure 2-7 Context Switch Profile

Headroom

Many distributed systems experience an "accelerating ramp" in response time as the load grows. Under light load, they exhibit very good-to-excellent response time, but this tends to slip as load is applied. Users often use perceived response time instead of utilization to understand how heavily loaded a system is. In some workloads, the loss of performance occurs at 50% or less utilization. This means that additional capacity is brought online at relatively low utilization.

Refer to Figure 2-8 on page 32 for an example of this concept. In this graph, the 250 MHz 32-way machine achieved 25% faster turnaround than an S/390 G4 2X 10-way sysplex on this group of batch jobs, employing 60% more processors at low utilization. However, as the load was doubled and then tripled, this advantage was not maintained.

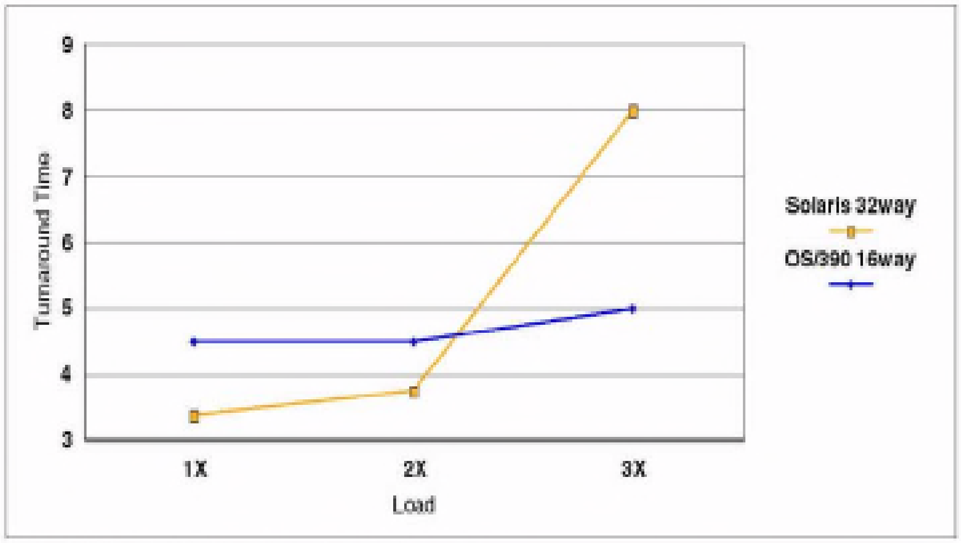


Figure 2-8 An SAS workload capacity curve

Redundancy

Many distributed solutions contain redundant systems as backups for availability, test, development, integration, etc. These extra non-production systems add capacity but are sometimes not used at all, or have very low utilization, or peak in utilization at different times than the production systems.

Fragmentation

Workloads grow as a continuum, whereas capacity is added in discrete quantities. As a result, whenever there is enough capacity configured, there is some amount of white space just based on the difference between the quantum step and the continuous growth in the workload. When the solution is distributed, the quanta are often whole systems, or the capacity is a set of individual sub-workloads, each continuously growing and having its demands met by its own quantum steps in capacity. This results in fragmentation of the resulting white space. By combining the loads, only one quantum of workload is only partially filled, instead of several.

Partitioning for integrity and isolation

Sometimes the user will create many small systems so that a failure will only impact a small subset of users; this is known as partitioning. Work is also partitioned, in order to create isolation and prevent interference. Finally, partitioning is often used in place of prioritization, so that each sub-workload gets the full attention of the machine upon which it is run.

The end result of any of these actions is white space. Partitioning also leads to the replication of data, often requiring off-shift batch updates which can drive higher peaks than the daily online workload. This leads to white space outside the batch window.

Skew

When multiple processors or systems are working in parallel, they are almost never kept uniformly busy. Much effort is expended in balancing workload, but in the end, none of the various methods for doing so work perfectly. The result is that white space emerges on some machines, when others are heavily loaded.

Regardless of the source, white space represents wasted compute power. Server consolidation is a means by which to remove some of the waste and run more efficiently.

2.4 Example sizing - analysis of company XYZ

Now that most of the theory for sizing has been discussed, let's take a look at sizing a given configuration.

This is the setup we inherit at the fictitious company XYZ.

Table 2-1 Setup for company XYZ

Function	Server type	# of servers	Average utilization
File Server	Compaq DL380	10	10%
DNS Server	Sun 5S	4	15%
Firewall	Sun420R	2	15%
Web Server	Sun280R	10	15%

Note: Before we go through each of the elements of sizing, keep in mind that many of the calculations we base our sizing on are confidential and cannot be explicitly written out. There are several reasons for this, the most important being we do not want to set a "standard" for how to size. Although this may seem counterintuitive, when one considers how many variations there can be in hardware (notice that our setup is fairly small and homogeneous, which will not always be the case), software, and workload, one can see why we cannot endorse a generic formula with some constants and a few variables. Since each situation is different, each sizing will have to vary accordingly. The intent here is to illustrate the principle, and not the specific implementation.

CPs

The engines of a system are always considered its primary attribute—even in the traditional mainframe environment (where emphasis on the processor is not as heavy as it is in the PC world). Accordingly, most of our efforts will be concentrated here.

The theory is deceptively simple: you measure how much load is on the current system (in other words, how heavily utilized it is), then translate this number into a zSeries equivalent via a workload factor (WLF), and that's that. The formula which represents this calculation is:

```
MIPS needed = %Utilization * Current Capacity * WLF
```

Making comparisons

As mentioned previously, architectural differences between processor families make comparisons between them very difficult. The clock speed competition between chip manufacturers has escalated recently, and this competition helps underscore the point. What makes it even more remarkable is that some of these chips share the same architecture. How much harder it is, then, to compare processors from completely different architectures.

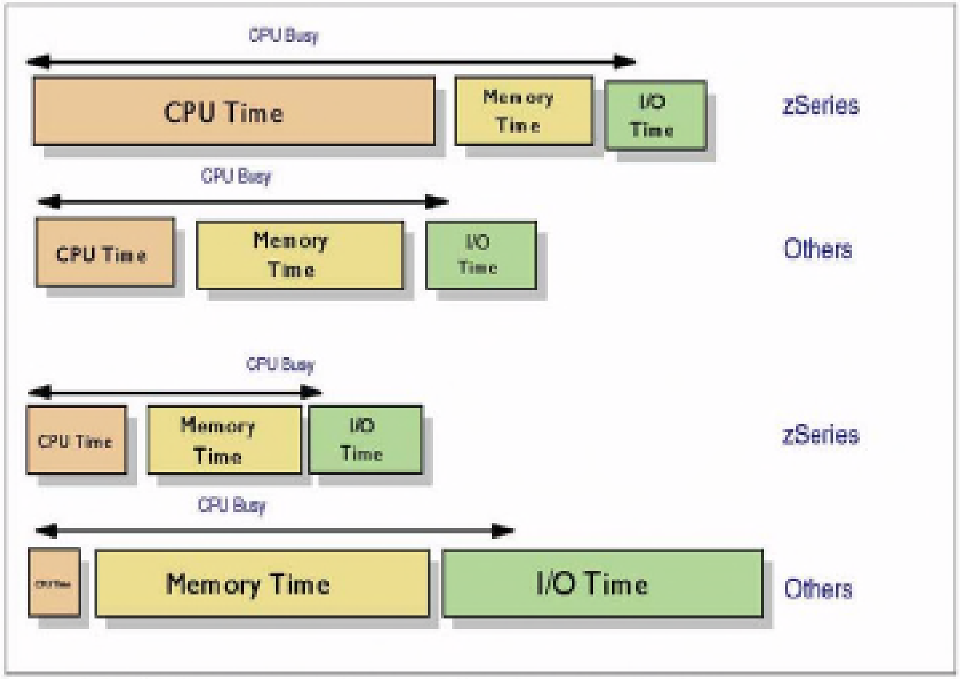


Figure 2-9 CPU-intense work (top) vs. data-intense work (bottom)

This is especially problematic for the S/390 and zSeries processors, since they are much better at data-intensive workloads such as databases, Business Intelligence, On-Line Transaction Processing, and "cache-killer" applications than they are at CPU-intensive tasks such as benchmarks and graphics rendering (see Example 2-9).

Nevertheless, that is the task before us now. Let's examine each component of the equation separately.

Utilization

Determining utilization is the most straightforward element. Since running Linux under VM means consolidating discrete, underutilized servers, it is important to find out the current usage. There are a variety of tools for the job, depending on the platform you're starting out on (Norton Systemworks for Windows servers, for instance). The key here is to have a duration that is long enough to give a realistic average utilization, and to identify what the peak time characteristics are, as explained in the previous section.

Remember that the more accurate the reading, the better off your sizing will be.

Current capacity

Determining current capacity should be fairly simple, as well. What we are referring to here is the TPC-C benchmark "transaction per minute" (tpm) numbers for each machine as they are configured. Obviously, the way these numbers reflect reality will vary somewhat depending on the workload, but they are good starting points.

If your workload is usual in some way, you are always free to compile your own data and come up with an unofficial tpm number, or even "tweak" the official numbers so they are more representative of your situation. Just be aware of the risks involved should these numbers be off.

Workload factor

The challenge here is to find a conversion factor to take us from our utilized tpm rating to the traditional S/390 MIPS rating.

Note: As already discussed, using MIPS numbers is not the best way to gauge performance for S/390 and zSeries machines. For performance measurements, LSPR is much better. However, keeping the end goal in mind, all we're after here is a conversion from tpm to MIPS so we can identify how many processors are needed to run the same workload as our starting environment on our target S/390 or zSeries machine. This is not to be taken as an endorsement of the MIPS rating in general.

Much easier said than done, of course; the pitfalls are many. However, we'll limit our discussion to the two primary issues:

- Usage & workload - Chances are you're extremely tired of seeing this by now, but the fact remains that you must consider the application and work that is being ported, in order to size it properly. This is not idle speculation we're engaging in. In terms of tpms, the relative capacity of a S/390 engine can vary from around 20 tpm/MIPS in CPU-heavy benchmarks to well over 200 tpm/MIPS for Samba workloads.
- "Generation Gap" - At the time of writing, there are three generations of IBM eServer zSeries processors we are looking at running Linux under: the G5, G6, and z900 engines. Needless to say, there are numerous differences between them. For instance, the IEEE floating point instructions are implemented in microcode in the G5 chips, but are done via actual circuitry in the G6 processors. The resulting performance increase is somewhere in the neighborhood of an order of magnitude. Therefore, the S/390 or zSeries processors you'll be migrating to is also an important consideration.

Memory

Also known as *storage* to the traditional mainframe folks, the zSeries machines are at a definite advantage when it comes to memory. Compared to its Intel and UNIX counterparts, the zSeries memory architecture is much more mature and efficient.

Consider Figure 2-10 on page 37. You can see the large breaks in the bandwidth as the L1 and then the L2 cache sizes are exceeded by the amount of data to be moved. Converting these results to tpm/MIPS yields the results shown in Figure 2-11 on page 37. Thus we can surmise that the tpm/MIPS grows with working set size, which drives cache misses and stressing the internal bandwidth shown here.

Refer to 3.3, "Memory topology" on page 55 for a more detailed analysis of this topic.

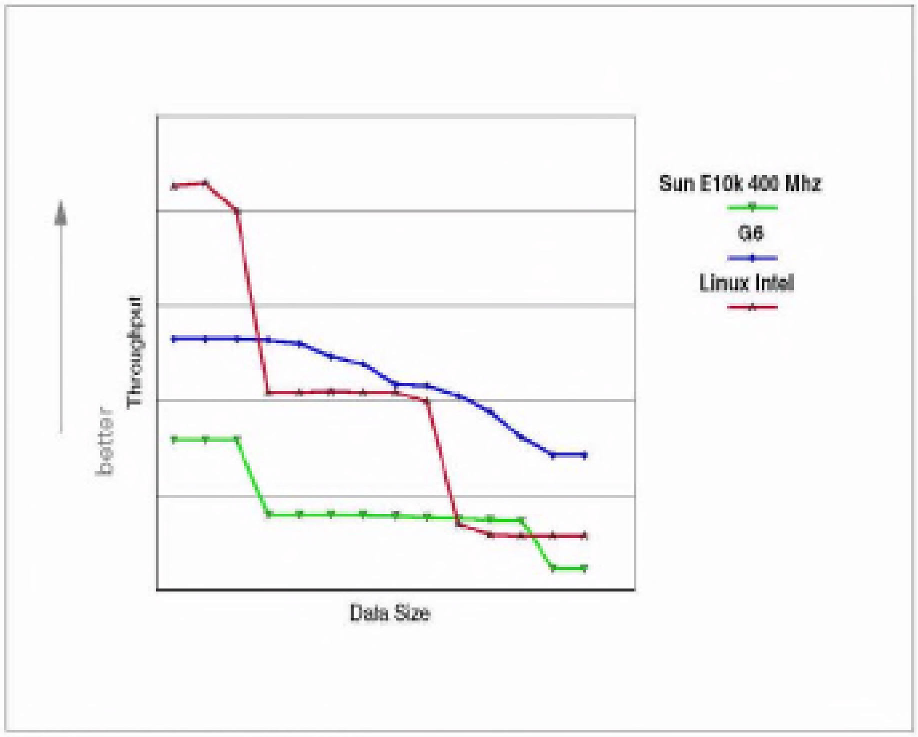


Figure 2-10 Memory read bandwidth

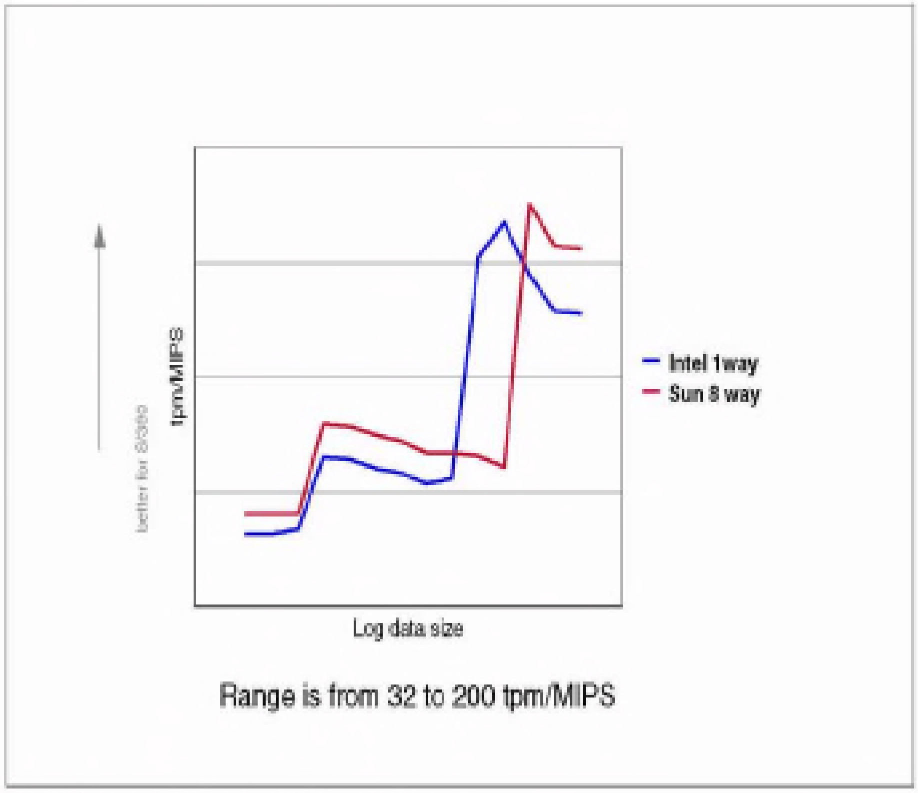


Figure 2-11 tpm/MIPS vs. working set size

DASD, channels and network bandwidth

I/O is heavily dependent on application characteristics. The situation on the source platform can be assessed using tools like Netbench, but a porting discipline has not really been developed for these factors yet. Refer to 3.2, “Disk topology” on page 46 and 3.4, “Network topology” on page 64 for a discussion of how these systems work.

2.5 Concluding remarks

In general, we are looking at a range of 30 to 200 tpm per MIPS of relative capacity. Data-intense workloads fall at the high end of this range, while CPU-bound workloads fall at the low end. The tpm per MIPS also can go up about 30% between a zSeries uniprocessor solution and a 16-way solution.

The art of sizing is in measuring the utilization properly, and choosing a WLF to use.

The TPC policies do not allow for the publication of unofficial TPC-C results or estimates, and IBM supports this policy. No single benchmark can accurately represent the way a system will perform in a specific customer environment. IBM sizing teams use multiple sources of information, including confidential estimates of benchmark results to deliver the best proposal possible for a given situation.

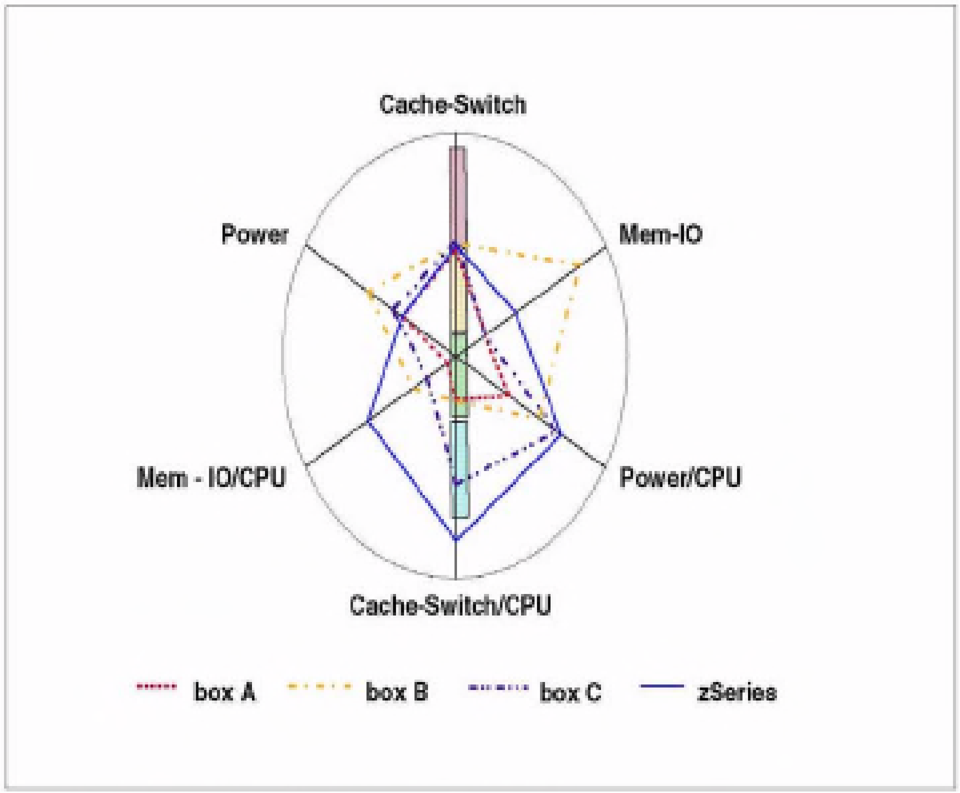


Figure 2-12 Capacity Profile - a more complete view

In Figure 2-12, you can see that box C is stronger on cache and internal bandwidth. You'll also notice that both boxes A and B are weak in "parallel hell" (for example, highly integrated transaction systems which requires significant sharing and synchronization which cannot be buried by redundancy) and strong in "parallel nirvana" (where the serial portion of the execution is small). The zSeries is ahead of the rest in parallel hell but weak in parallel nirvana, particularly if the work is CPU-bound. Box A is a relatively small machine when looked at this way.

2.5.1 Total Cost of Ownership (TCO)

Now that we've established how to quantify relative capacity, we need to understand how to compare costs between servers. It is not enough to tally price/performance, but rather it is necessary to understand the total cost associated with each server type. The reason for this is that in raw capacity, you can always put together enough small machines with low prices to show a better price and capacity advantage for a distributed solution. Therefore, simply looking at $ per tpm as suggested by the published TPC-C benchmark results will be misleading, even if the tpm/MIPS (or tpmA/tpmB for that matter) is adjusted to account for different architectures.

This is because differences in server architectures and implementation go beyond capacity. Even *within* a family of servers, differences between clusters and a large machines appear. These variables drive differences in a variety of non-acquisition costs, such as occupancy, network engineering, operations (people), and outage costs. Therefore, to understand which server type should be deployed, it is necessary to look at the Total Cost of Ownership (TCO) of competing solutions.

However, even this is not a simple matter of adding up well-known average values to get a "typical" result. The problem is that there is a very large variance in each of the variables involved. There are usually unique and significant cost factors which can only be listed as "other costs" in any one-size-fits-all method of computing the TCO. Having said that, the following is an attempt to build a TCO model which fits most situations.

The Total Cost components

Here is an outline of the components of Total Cost:

1. Time
   a. How far into future
   b. How often are upgrades
   c. How many upgrade steps in Study
2. Hardware
   a. Price
   b. Discount
   c. Maintenance
   d. Financing
   e. Overlap, Deployment, or Depreciation cost
   f. Node Count
   g. Rack Count
3. Software
   a. Price
      i. Per seat
      ii. Per CPU
      iii. Per Unit of Capacity
      iv. Per Box
      v. One time or Monthly License Fee
      vi. Cost per Rack (includes power connection)
   b. Maintenance
4. Occupancy
   a. Burden per SQ ft. (Rent, facilities, lights, heat, cooling, etc.)
      i. Rent
      ii. Facilities
      iii. Lighting
      iv. Heat/Cooling

        v. Clearances (Sq. foot per Rack)
    b. Power
        i. $ Per Kilowatt Hour
        ii. Kilowatt Hour per Rack (estimated)
5. Storage
    a. Total Bytes
    b. Compressed Bytes
    c. Redundant Bytes
    d. Tape Drives
    e. Replicated Data (does the solution consolidate it?)
    f. SAN, Integrated or External implementation (or apportionment)
6. Network
    a. Cost per Connection
    b. Routers per Rack
7. People
    a. Operational
        i. Uncloned
        ii. Cloned
        iii. Super Cloned
    b. Skills
        i. Per Architecture
        ii. Servers per Administrator
    c. Automation investment
        i. Current Investment
        ii. To be developed
8. Outage Costs - Loss Model
    a. Productivity Loss
        i. User Count
        ii. % Users effected
        iii. User Burden Rate
    b. Revenue Loss
        i. $/Minute of opportunity losses
        ii. $/Minute of penalty losses
    c. Loss of Confidence or Reputation (Stock Price loss)
        i. Publicity exposure
        ii. Stock price vulnerability
9. Other costs
    a. Migration Costs (Databases, Middleware, ISV code, etc.
    b. Porting Costs (Home Grown Applications)
    c. Facilities engineering costs
    d. Network Engineering costs
    e. Solution Architecture costs
    f. Reengineering for scalability costs

2.5.2 Some trade-offs

With sizing, there is a fundamental decision to be made up front: do you upgrade, replace, or add to the existing infrastructure? Each of these paths leads to a different set of trade-offs in TCO.

For example, a customer may need to double its capacity. If the customer decides to upgrade existing boxes, this is typically done by adding processors, memory and I/O to each box. In this case, overlap costs are small and network engineering costs are small to large, depending on the whether network connections are added to the existing boxes. Staffing can remain flat, power is up marginally, and floorspace only goes up if routers are added. However, there is the risk that scalability engineering costs occur, if the applications reach internal scaling limits on the upgraded systems.

Alternatively, suppose the customer decides to replace hardware. Now there are depreciation costs and deployment costs to consider. Network, staff/operations, and floorspace costs may potentially go down, but one is still faced with the same scalability engineering issues (more work per instance).

Finally, let's assume that the customer decides to add hardware. In this case, floorspace, people/operations, network and other costs go up, but scalability engineering issues are different (for examplehow parallel is the workload, how good is the load balancing, and how good is the data partitioning). Now, assuming we are doubling the boxes to double the capacity, this typically means that utilization on the new boxes is lower than on the existing boxes. Alternatively, we can chose to add fewer, but larger boxes, in which case the scalability engineering requirements would include both the larger instance and more instances forms of scaling. In this case there is little or no depreciation, but there may be some overlap cost.

Another fundamental decision is the choice of upgrade granularity, which is how frequently will capacity be added, or in what increments.

2.5.3 Final reminder

We've stated several times in this chapter that sizing is an important but difficult process. In our experience, even if you were to consult sizing experts on the best methodology (especially for Linux on the zSeries and S/390), chances are that you'd end up with as many methods (and results) as there were experts.

As a matter of fact, a Gartner Research Note (P-13-7373) released in June 2001 states:

```
There is no easy way of initially sizing how many MIPS an S/390 or zSeries
will require to handle projected loads, especially with the varying system
utilization of a large number of servers.
```

However, with the proper preparation and an understanding of the topic, a good sizing can be done. This chapter should serve as a guide to that understanding, as well as a bridge to resources that will help you along the process. It may be helpful to refer again to 2.2.2, "The bottom line" on page 25. While neither simple nor straightforward, this is an important phase of any migration project which cannot simply be ignored.

3

Virtual server architecture

In this chapter we discuss aspects of running many “virtual” servers on a single mainframe. We do not address the clustering of virtual servers, but instead examine the issues involved in structuring a large collection of servers for manageability and efficiency. At the end of the chapter, we offer recommendations based on our testing.

The chapter introduces concepts that are expanded upon in later chapters.

3.1 Why an architecture is required

In large, discrete server environments (and even in some smaller ones), there are a number of management and operational issues to be faced. These include:

Scalability	How to allow for increasing capacity in the environment
Management	How to control, configure and monitor the environment
Growth	How to add new servers to the environment

The penguin colony provides solutions to some of these issues, which one reason why the concept is attractive. However, if the structure of the environment is not planned in advance, the benefits of the solutions provided by the penguin colony are reduced.

By designing the penguin colony according to a set architecture (what we refer to as a" virtual server architecture"), you can build the environment so that issues of scalability and management are addressed, and so that it becomes easy to create new servers for the environment when required.

3.1.1 Components of the virtual server architecture

Here we introduce the components of the virtual server architecture, and discuss alternatives for their implementation. The components that must be considered in the architecture are:

- Disk topology - see 3.2, "Disk topology"
- Memory topology - see 3.3, "Memory topology" on page 55
- Network topology - see 3.4, "Network topology" on page 64

At the end of this chapter, we offer recommendations based on the testing we performed. However, different approaches may suit your environment better than the choices we made in our testing. The purpose of this chapter is to present a discussion of the issues, so that you can decide on an architecture which works for your installation; in the language of Internet newsgroups, YMWV[1].

3.2 Disk topology

This section discusses the ways that disk can be allocated to Linux images in a penguin colony.

[1] Your Mileage Will Vary, meaning your experiences will almost certainly differ from ours.

3.2.1 The DASD driver

Linux disk support is provided by the DASD driver code, dasd.c. It provides support for Count-Key-Data (CKD) and Fixed Block Address (FBA) disk devices, as well as VM minidisks.

In the Linux installation systems, the DASD driver is provided as a module. This is because the disk configuration is not known, and is determined as part of the installation. The installed system has the DASD code built in to the kernel, and parameters to the DASD driver are passed as part of the kernel parameter line.

The DASD driver uses channel I/O to perform read and write operations. For VM minidisks using the CMS RESERVE format, VM Diagnose I/O can be used to provide better performance. More detail on this can be found in 8.10.1, "VM Diagnose I/O" on page 163.

3.2.2 Linux instances with dedicated disk

The usual method of installing Linux is to dedicate disk volumes to each instance. In this scenario, every Linux instance has its own disk volumes that no other instance has physical access to, and each instance is installed in the same way.

This is not a very efficient installation method for a penguin colony, as there will be a large amount of redundant data kept in each instance—much of which will be the Linux installation itself. Software management in this scenario is intensive, because each Linux instance maintains its own copy of all applications, from user-level software right down to the kernel.

Also, because there is currently no way to partition DASDs at the Linux level, there is very little granularity in the possible DASD allocation (however, the command `fdasd` will be coming to the Linux for zSeries and S/390 distributions).

This problem can be addressed in two ways:

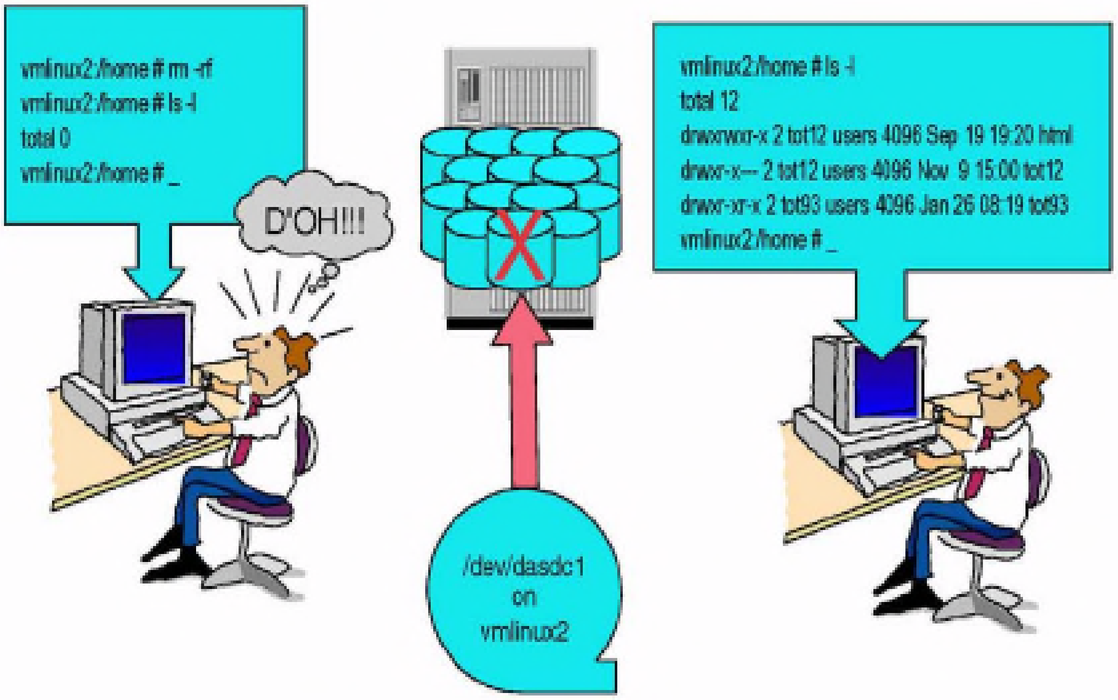
- At the Linux level, you can use LVM to create virtual volumes that may span multiple physical DASDs (at the expense of a longer code path to the actual disk).
- Using VM, minidisks provide a solution by dividing a physical volume into multiple virtual disks at the VM level.

Either of these options allows the equivalent of partitioning.

The benefit of this approach is its isolation. In some environments, the ability to create Linux instances that are *entirely* separate from each other is very attractive, and worth the management and definition overhead. By keeping every disk separate from all others, administrators are free to treat the instance just like they would a discrete server; they can install their own software, maintain their own configuration, and run their own services.

However, in spite of the operational isolation of this approach, you can still use VM concepts to improve management over a discrete server approach. Providing disaster recovery facilities in this scenario might be as simple as taking copies of the minidisks allocated to Linux machines, and backing them up. Restoration of a failed system would then be a matter of restoring the disk images and booting up.

This approach could also be taken with massive data loss, as Figure 3-1 illustrates.



```
vmlinux2:/home # rm -rf
vmlinux2:/home # ls -l
total 0
vmlinux2:/home # _
```

```
vmlinux2:/home # ls -l
total 12
drwxrwxr-x 2 tot12 users 4096 Sep 19 19:20 html
drwxr-x--- 2 tot12 users 4096 Nov  9 15:00 tot12
drwxr-xr-x 2 tot93 users 4096 Jan 26 08:19 tot93
vmlinux2:/home # _
```

Figure 3-1 Simplified full-volume recovery scenario

In this example, a utility such as VM DDR is used to replace the entire disk image. Done simplistically, as shown, this approach is non-selective (i.e. it does not provide a means of retrieving portions of the file system), so it is a method that would suit the recovery of *entire* file systems rather than individual files in a file system.

More discussion on disaster recovery and backup scenarios appears in Chapter 7, “Backup and restore” on page 127.

3.2.3 Linux sharing data

Because many members of the penguin colony will be identical in software configuration, it would be ideal to have these instances share this common data and programs (for example, this might apply to a group of Linux instances that provide the same service or application to a particular customer). The data relating to the service must be shared between the servers for consistency.

At the same time, there will be certain data that will have to be different from one instance to another (such as network configuration data). This data can be stored on a disk device unique to each instance, but management of this system-unique data would be simplified if it could be accessible from a single location (or from all locations) in the penguin colony.

Since the members of our penguin colony have virtual network connections between them, we can use facilities that make data available across a network. There are at least two ways to do this:

- “Traditional” network file sharing

  A facility such as Server Message Block (SMB) or Network File System (NFS) is used to provide discrete file system access between servers.

- Global namespace

  A virtual file system which introduces a global name space to its members servers, such as AFS or GFS, provides universal access to data across the penguin colony.

Server Message Block (SMB)

Also known as the Common Internet File System (CIFS), SMB is the file sharing protocol used by operating systems like Microsoft Windows and IBM OS/2. Linux supports SMB through the Samba application suite.

In the situation where you want to share the data on your penguin colony with SMB clients (such as Windows desktops), running Samba on your Linux machines is the best approach. However, using Samba to share between Linux instances does not work well, mostly because Samba does not maintain the UNIX security information across the network. See Chapter 15, “Integrating and replacing Microsoft servers” on page 345, for hands-on details.

Note: This does not mean that Samba cannot be used between Linux machines! For simple file access it is quite capable, especially where a Linux machine is set up as a Samba server. In this case, using **`smbmount`** or **`smbclient`** from another Linux machine to perform ad hoc sharing is easy. For significant sharing of data between Linux systems, however, a “native” method that observes the Linux security model is a better choice.

Network File System (NFS)

NFS is a common method of sharing file system data among UNIX platforms. An NFS server makes parts of its file system available over the network, allowing NFS clients to mount the served file system data into its own file structure. Also, NFS observes the Linux/UNIX file permission model, so file permissions can be managed uniformly across an installation.

NFS does have limitations; in this discussion the most important consideration is an inherent lack of security based on its use of Remote Procedure Call (RPC). RPC, which uses UDP as the transport mechanism, does not provide a high level of security for two reasons:

- UDP packets can easily be “spoofed”, defeating security based on the IP address (or host name) of the source machine.
- Authentication is performed “in the clear” over the network.

Note: A complete discussion of NFS security is beyond the scope of this book. Most Linux security books offer information on NFS security.

One way to confidently use NFS in a penguin colony is to restrict NFS traffic to separate network connections, isolating it from other traffic; see Figure 3-2 on page 51. This obviously increases both management and processing overhead.

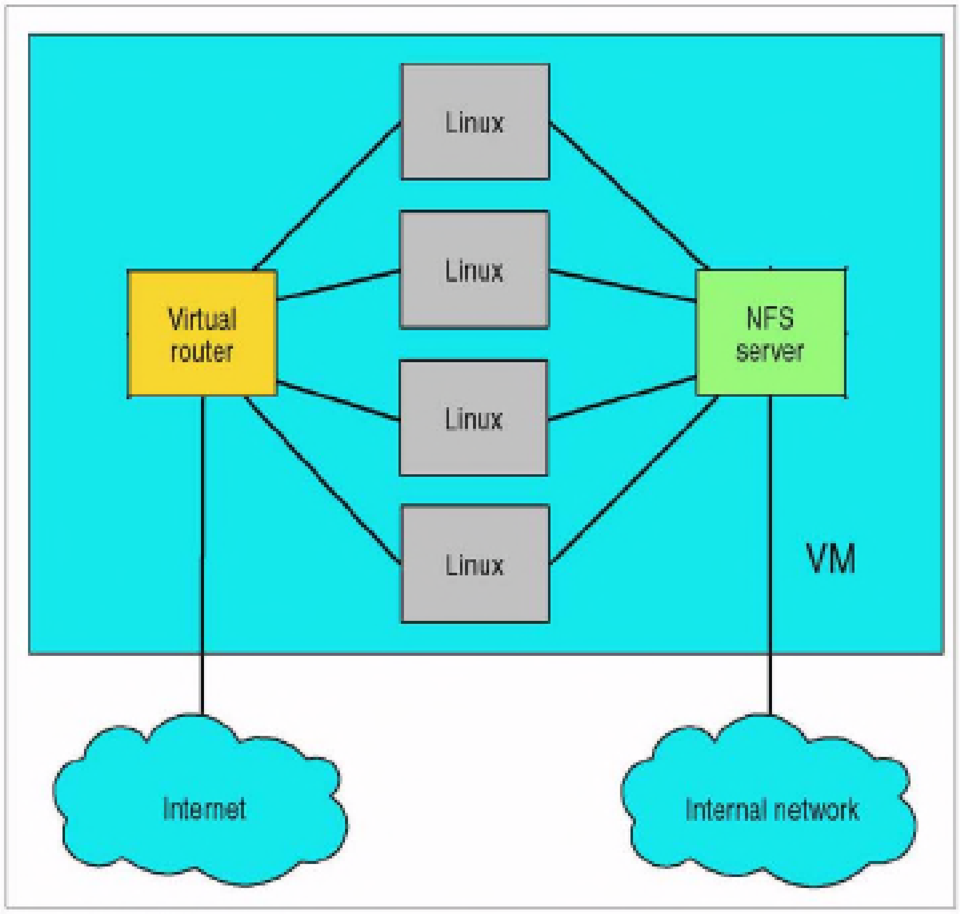


Figure 3-2 NFS isolation

The design of NFS does not automatically lend itself to providing a global namespace, but in theory a single NFS client could mount file systems from many NFS servers, which would provide a location where the relevant parts of all file systems could be accessed. However, as the number of instances increases, this arrangement would be very intensive to administer unless a utility such as `automount` were used to keep track of the mounted file systems.

Global namespace

The concept of a global namespace brings together the file systems of many separate servers into a single structure. This happens seamlessly to each member of the name space.

Global File System (GFS)

GFS is a cluster file system that provides a common name space to member servers. All servers in the GFS structure can access these physical disks. Disk devices are allocated to storage *subpools*, providing the flexibility to group disk devices according to performance attributes. Then, the file system can physically locate files in an appropriate subpool according to performance requirements.

To arbitrate access to the shared disks, either device-level locking is performed, or a global locking server is introduced. On platforms that support it, this locking server uses shared memory for data structures; otherwise, network communication over TCP/IP is used.

In the Linux-VM case, shared access to the physical disk can be provided by linking physical disks (or minidisks) to all of the systems participating in the GFS pool. It is also possible to use the network block device, mounting the physical disks to one system only and accessing the disk via the network.

Figure 3-3 illustrates the various GFS scenarios.

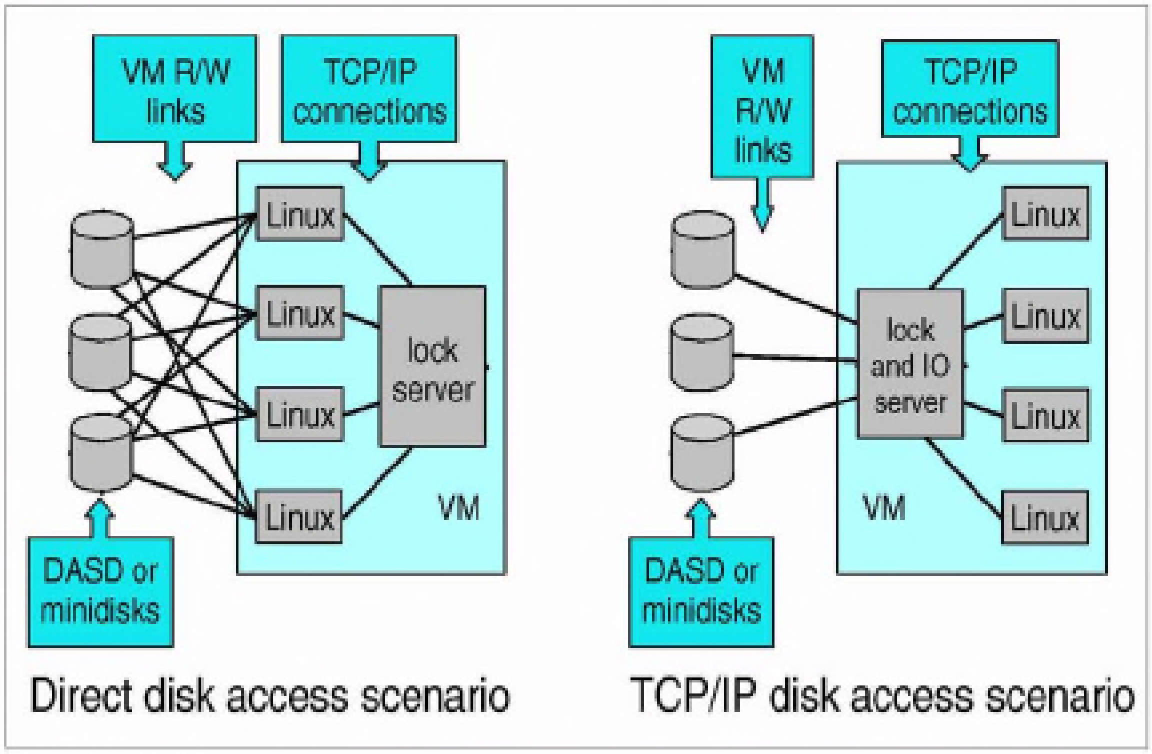


Figure 3-3 GFS scenarios

Another advantage of GFS is that it is a journalling file system, providing better file system integrity and faster recovery after a system failure.

The global name space is implemented (in part) with context-dependent symbolic links (CDSLs). CDSLs are generated by GFS to integrate certain system-specific file system components (such as /etc) into the global name space.

At present, support for GFS in Linux on zSeries is in trial stages. There are difficulties in compiling it on 2.2 kernels, and even on the 2.4 kernels it is not well proven.

Andrew File System (AFS)

AFS is another implementation of a global name space. In addition to the seamless file system, AFS provides much better security than NFS.

An Open Source implementation of AFS can be found in OpenAFS, the latest version of which is 1.0.4a. While AFS is well proven as a commercial product, OpenAFS has yet to prove itself. However, it is reasonably assured of success given the large number of organizations with experience in AFS. See 15.4, "Using AFS in an enterprise environment" on page 362 for a hands-on description.

The Coda file system

Coda (for common data access) is a descendant of AFS, designed by Carnegie-Mellon University. Coda provides similar features to AFS, with the addition of a "disconnected" mode of operation, where clients can work on files in their own local cache if the server is unavailable. When the server is contactable again, the cache is synchronized automatically. This aspect of its design indicates its primary use as an accessible file server for end-user data, rather than a shared file system for servers. More information on Coda can be found at the Coda Web site:

`http://www.coda.cs.cmu.edu`

One advantage of Coda is that the Linux kernel supports it natively. However, its development status is unclear (the bug tracking system is active, but the FAQs don't appear to be up to date).

3.2.4 Sharing disk in memory

In 17.3.3, "Using shared segments" on page 411, we describe our work in developing a way of using the discontiguous saved segment (DCSS) capability of VM to create an in-memory virtual disk which can be shared across multiple members of the penguin colony.

This is a very promising area, which would have great benefits to large Linux virtual server environments. For example, it would be an ideal way to provide very fast read-only access to the /usr directory for a penguin colony, not only increasing performance but simplifying software management as well.

3.2.5 Minidisk caching

VM provides a feature that can provide a good performance benefit for physical disk access, which will assist in any of the scenarios described here. VM minidisk caching, as the name suggests, allocates memory in VM to cache guest minidisks to accelerate disk access.

While Linux provides its own buffer cache, it is still advantageous to provide a "second-level" disk cache, because the Linux cache takes lower priority to real user processes in Linux's memory allocation. This means that there may not always be enough free memory available for Linux to provide an effective cache internally. The memory for the minidisk cache is preallocated, so there is always at least one level of caching available to the Linux guests.

Minidisk caching has been shown to dramatically improve the performance of Linux guests with file systems on minidisk. There is a tradeoff, however, because allocating memory to minidisk cache means that there is less memory available to be allocated to guests. Adding more memory to the processor may be justified in order to be able to take advantage of the available performance gain.

Minidisk caching can be done at a track level or at a block level. Refer to 8.10.2, "DASD MDC measurement" on page 166 for an analysis of these options.

3.2.6 Limitations of sharing disk

While it is desirable to share as much common information as possible, the sharing disk approach introduces issues that must be managed.

Caching

If a number of Linux instances have shared access to disk, they will each be holding a copy of disk blocks in buffer cache. Usually this is not a problem, but if one of these Linux instances writes to the disk, there will be a period of time where the other systems sharing that data have an inconsistency. For some applications, this may be intolerable. Therefore, the shared disk access method you choose must allow for this.

Also, because main memory is more expensive than disk, it may not be desirable to have many Linux instances holding copies of the same data in their own localized buffer caches. This would be particularly relevant when the file system is held in memory anyway, such as with VM Virtual Disk (VDISK).

The way that the Linux kernel is designed makes it very difficult to "turn off" the buffer cache. However, in the S/390 environment, in certain circumstances this would be desirable. The considerable amount of work that this would entail might be justified for certain configurations.

Locking

While discussing GFS, we discussed the need for a lock manager to arbitrate access to shared media. This requirement will vary, depending on the importance of the data and the way in which Linux instances access it. For example, a file system carrying HTML files mounted read-only by a number of Web-serving Linux instances arguably does not require a lock server (because none of the instances can alter the data).

3.3 Memory topology

The allocation of memory in a penguin colony is critical to the performance of individual instances, and of the entire installation. VM provides ways to improve the basic operation of Linux instances on S/390 and zSeries.

3.3.1 Linux 'jiffies'

The Linux kernel employs a 100 Hz timer that "wakes up" the kernel to check for new work. The 10-millisecond units of time between timer pops are generally referred to as *jiffies*, after the name of the kernel variable where the timer count is stored. Parts of the kernel (and some other software) use this timer to check for new work when the timer pops, every 10 ms.

To VM, the constant popping of the jiffies timer means that the Linux guest is always busy. This is because the idle detection processing in VM requires a longer idle time than the jiffies timer provides. This affects the way in which VM pages Linux's memory, because VM pages more aggressively on a guest that is idle than on a guest that is active. Also, the processing done by Linux at each of these timer pops becomes significant with large numbers of Linux instances, causing CPU constraints.

Note: The CPU impact caused by the timer results some confusion.

If the Linux instance was not processing at the time of the timer pop, it is driven to check for work, and this costs cycles. If the Linux instance was doing "real work" at the timer pop, that processing would be interrupted to allow the timer pop to be handled. Again, this costs cycles.

Therefore, in the penguin colony, reducing (or eliminating) timer processing has benefits regardless of the average load of the Linux instances.

The 100 Hz timer causes the most significant issue in management of memory in a penguin colony. The alternatives discussed in this section all provide ways to manage memory, but all are affected somehow by the 100 Hz timer. A patch to the Linux kernel has been written by IBM Boeblingen, which provides a kernel configuration option that removes the 100 Hz timer and replaces it with a different scheduling mechanism. Unfortunately, Linux developers often use the jiffies variable for general timing purposes (which sometimes have nothing to do with work scheduling), so simply removing jiffies from the kernel could have unpredictable results on other software outside the kernel.

IBM's patch is still regarded as very experimental and has not been widely tested. However, because other areas of the Linux community have shown interest in the 100 Hz timer issue (the user-mode-linux developers, for example), the patch has a lot of potential.

Information: We had access to a beta build of SuSE Enterprise Server 7.2 for S/390 and zSeries, which was released on July 3, 2001 and is based on the 2.4.5 kernel. However, even this build, created well after the original availability of the patch, does not have the patch applied. This reflects how experimental the patch is.

At the time of writing, the members of the Linux-390 mailing list were discussing the merits of making it part of the IBM "s390/s390x" patches to the 2.4 kernel.

3.3.2 Large guest memory

The simplest configuration, and the one which most closely mirrors running Linux on discrete servers, is to simply allocate to each VM guest the same or similar amount of memory as the discrete server would have.

Due to the way in which Linux utilizes RAM, however, this is not efficient in a penguin colony. The Linux kernel allocates unused RAM as buffer cache to improve disk performance. To VM, there are no unused areas of memory, which makes it difficult to determine what can be paged out; refer to Figure 3-4.

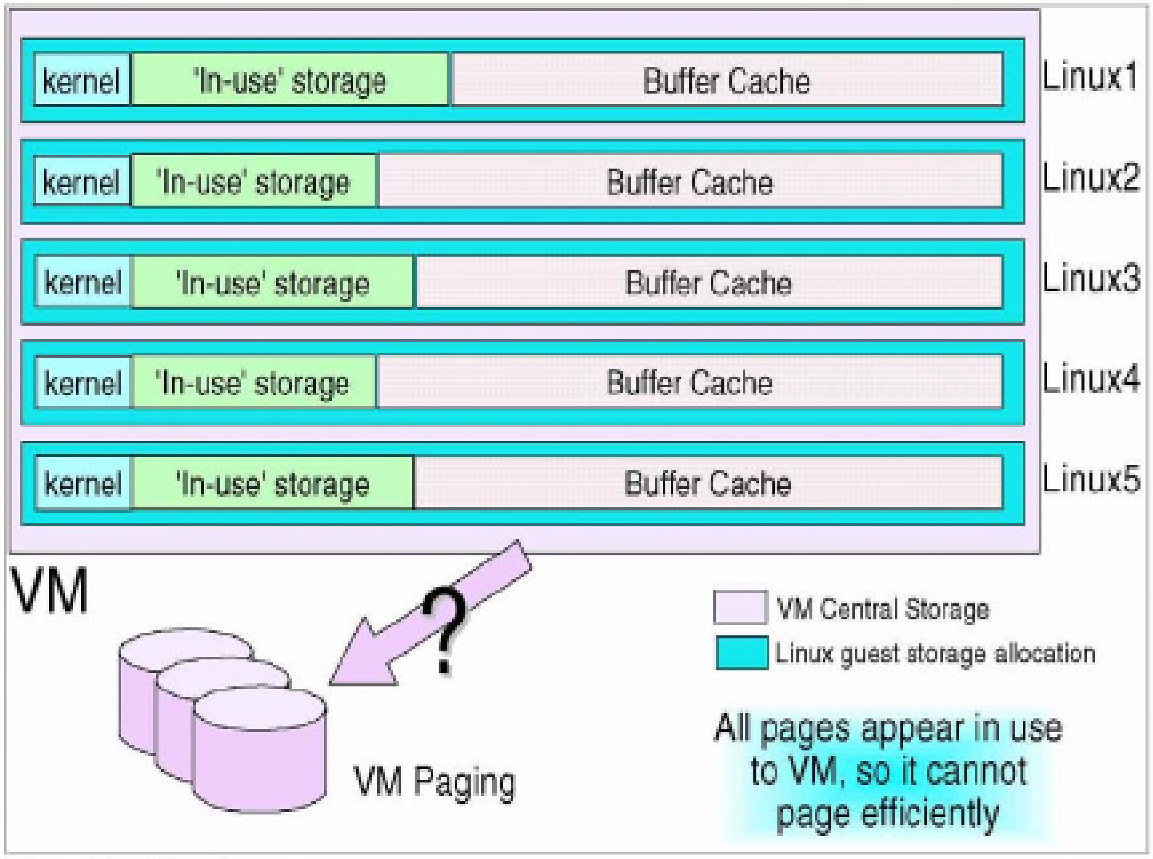


Figure 3-4 VM paging

Not shown in Figure 3-4 is the effect of Linux swap space, which further confuses the situation. Consider a memory-constrained Linux guest, with Linux swap space defined, which is operating in a memory-constrained VM system. VM will be paging Linux memory to relieve VM demands, but at the same time Linux will be swapping to relieve its own constraints. This can lead to a situation where pages of memory—marked as swapped-in by Linux but paged out by VM—are paged-in by VM simply to allow Linux to swap them out! This is generally referred to as *double-paging*, and is very undesirable.

An ideal solution allows VM and Linux to work together—or at least not fight each other.

3.3.3 Linux swap to VM virtual disk

An alternative way of allocating memory is to reduce the size of the Linux virtual machine, and allocate more swap space to it. In a discrete server environment, where disk access is orders of magnitude slower than RAM, this would be undesirable. Using VM Virtual Disk (VDISK) as Linux swap, however, we can greatly enhance the performance of Linux swap; refer to Figure 3-5.

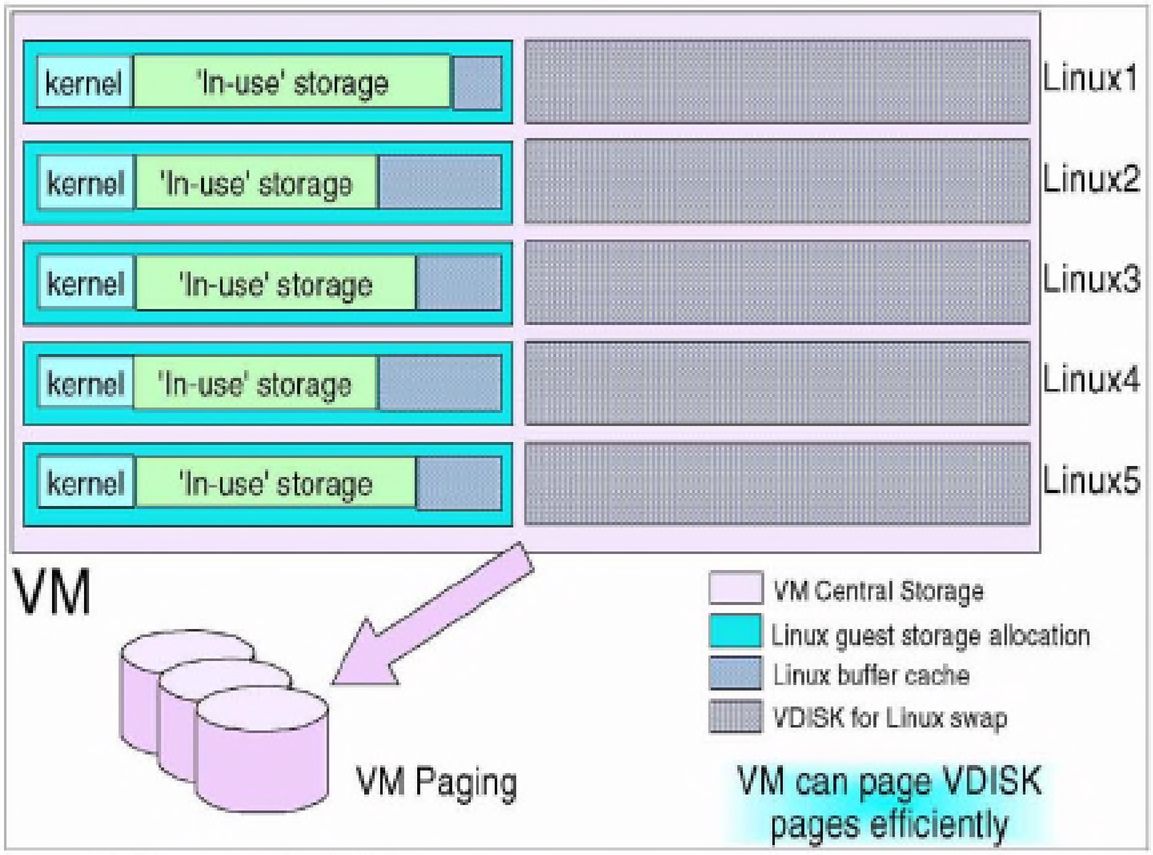


Figure 3-5 Linux swap using VDISK

In this case, even though in total we allocate a similar amount of memory to each Linux instance (once the amount of real memory and VDISK is added up), VM is able to manage the memory more efficiently.

However, there are limitations to this option:

- Applications which require a large amount of "real RAM" may not be able to obtain enough real memory to run[2].

[2] Empirical evidence suggests that WebSphere falls into this category, requiring a fairly large amount of core (in excess of 128 MB) just to start.

- Applications that are disk-intensive, and would benefit from Linux buffer cache, may experience slightly degraded performance (it is not as severe as forcing a disk I/O in every case if VM caching is in effect).
- Since Linux believes it is using a real disk, the DASD driver will still be used for swap I/O. This adds a slight performance overhead compared to memory access.

The VM paging constraint introduced by the Linux 100 Hz timer will still be an issue here, but because the size of the Linux guests' memory is reduced, the total amount of memory affected by the issue is reduced.

3.3.4 Linux swap files in memory

In order to further improve the performance of Linux's "virtual swap", we can take the somewhat radical step of having Linux swap into memory. There are two ways to achieve this, through using expanded storage or by using Linux RAMdisks.

Expanded storage

Linux can utilize expanded storage as a block device, using the XPRAM driver. This would allow an ext2 file system to be created in expanded storage, and Linux swap files to be defined there. This approach provides a fast swap facility, but there is little advantage between this approach and just allocating all of the memory to Linux as central. In addition, expanded storage is not supported on the z900 in 64-bit z/Architecture mode, so there would appear to be little future in this method.

Note: To clarify this point, z/VM supports expanded storage in 64-bit mode, but only for *its own use*. An operating system running as a guest in z/VM cannot use expanded storage. However, a guest OS could indirectly utilize z/VM's expanded storage through minidisk caching, VDISK or other z/VM services, if z/VM was configured to provide these services using expanded storage.

Linux RAMdisk

In this configuration, we increase the amount of memory allocated to the guest, but instead of using external swap devices, we allocate RAMdisks in Linux's memory and create swap files on these RAMdisks.

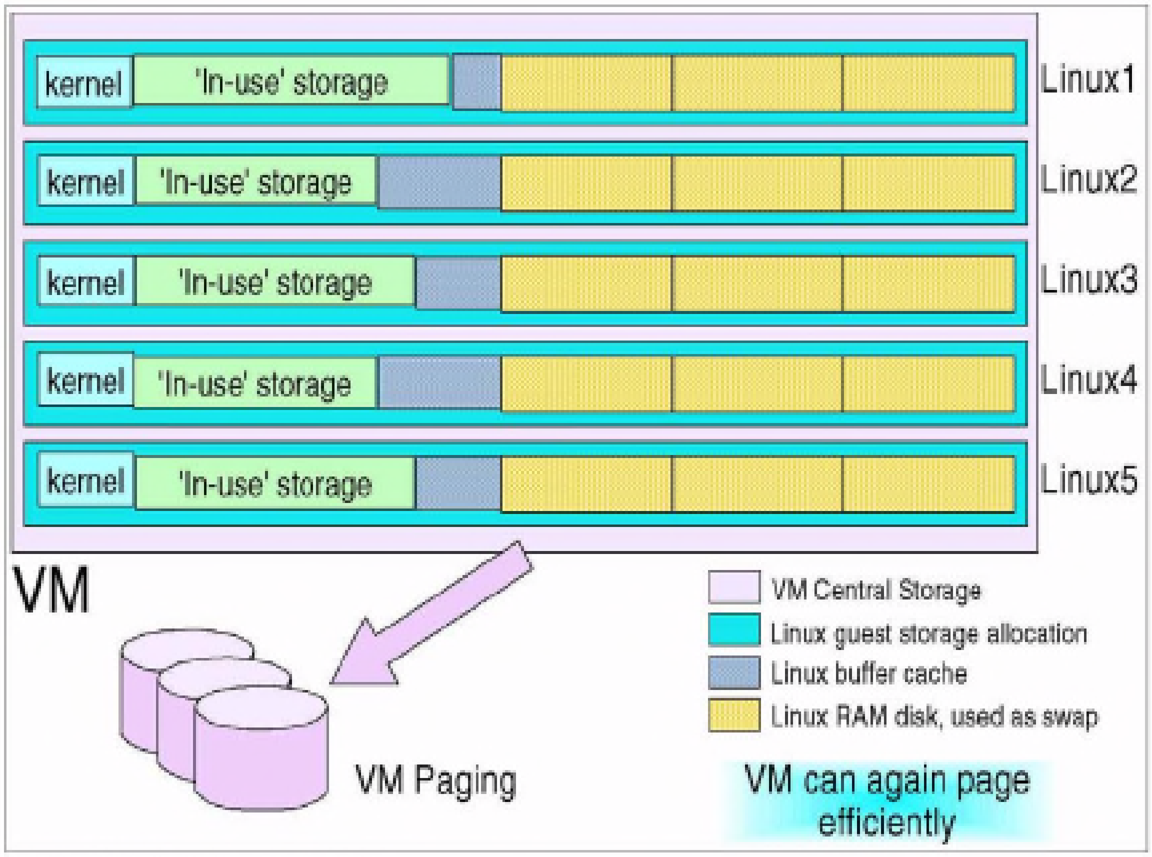


Figure 3-6 Linux swap to RAMdisk

This provides a number of advantages over the VDISK method:

- DASD driver is not used to access RAMdisk, thus reducing I/O overhead.
- Multiple RAMdisks can be set up, thus allowing good granularity in balancing memory available to Linux.
- VM can page the memory used by the RAMdisks more efficiently than if Linux was using it as buffer cache.

This configuration gives significant control over memory usage. For example, if an application requiring a large amount of "in-core" memory is started, RAMdisk swap files can be turned off and the RAMdisk deleted to return the memory to Linux (this is illustrated in Figure 3-7 on page 61).

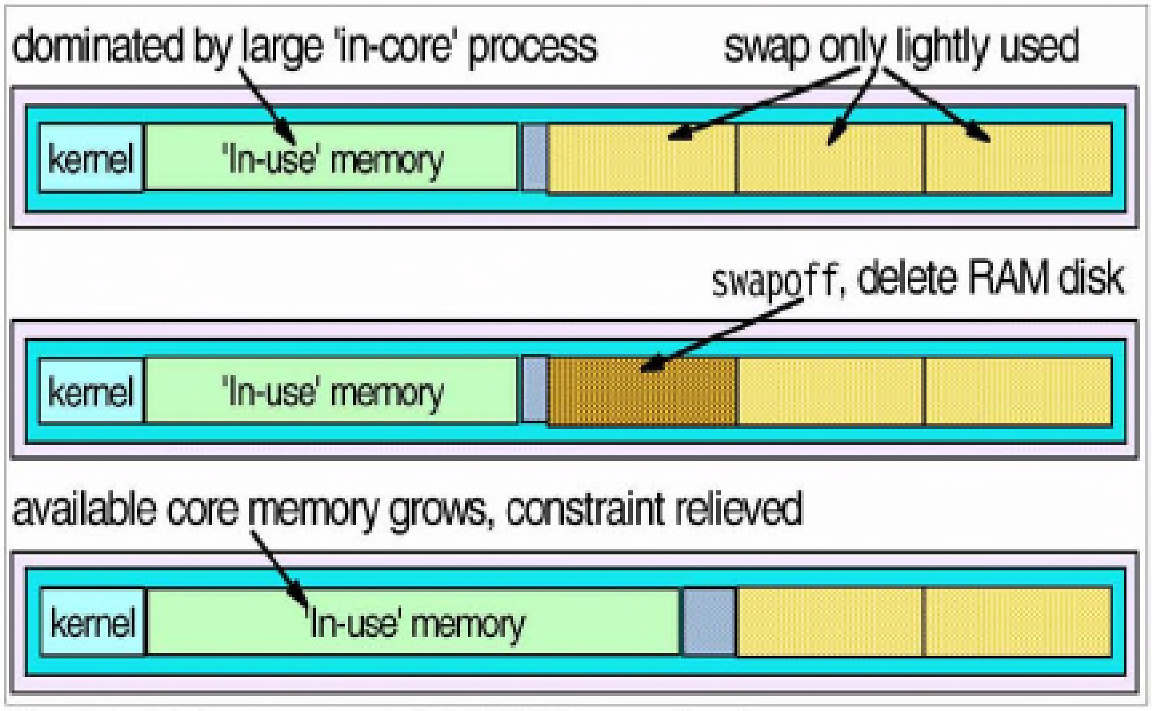


Figure 3-7 Relieving memory constraint in RAMdisk swap configuration

Note: The operation would only give benefits when the overall memory requirement was moderate or low, and a certain application required a large amount of memory "in-core".

The reallocation of swap RAMdisk in this way would not improve the operation of a Linux image whose total memory requirement was close to or exceeded the size of the VM. If this is the case, removing the swap RAMdisk would actually cause severe problems, because *all* the available swap space would be in use. To perform the operation, additional DASD swap space would have to be brought on temporarily to manage the change.

The downside of this method is related to the `jiffies` timer: by increasing the size of the Linux guest, we've reintroduced the larger impact of the timer on VM paging. For this reason, this method would work best with a kernel modified to remove the timer.

3.3.5 "Hybrid" swap method

By using both Linux RAMdisk swap areas and VDISK swap space, it is possible to create an extremely tunable memory architecture which can provide all of the benefits of the methods discussed so far.

In this design, we allocate a number of Linux RAMdisks—but only build swap space in *one* of them. We fill the others with large files to force Linux to allocate the memory to the RAMdisk. In addition, we allocate a VDISK and make Linux swap to that. The swap space on the VDISK is marked as lower priority than the RAMdisk swap space. This is illustrated in Figure 3-8.

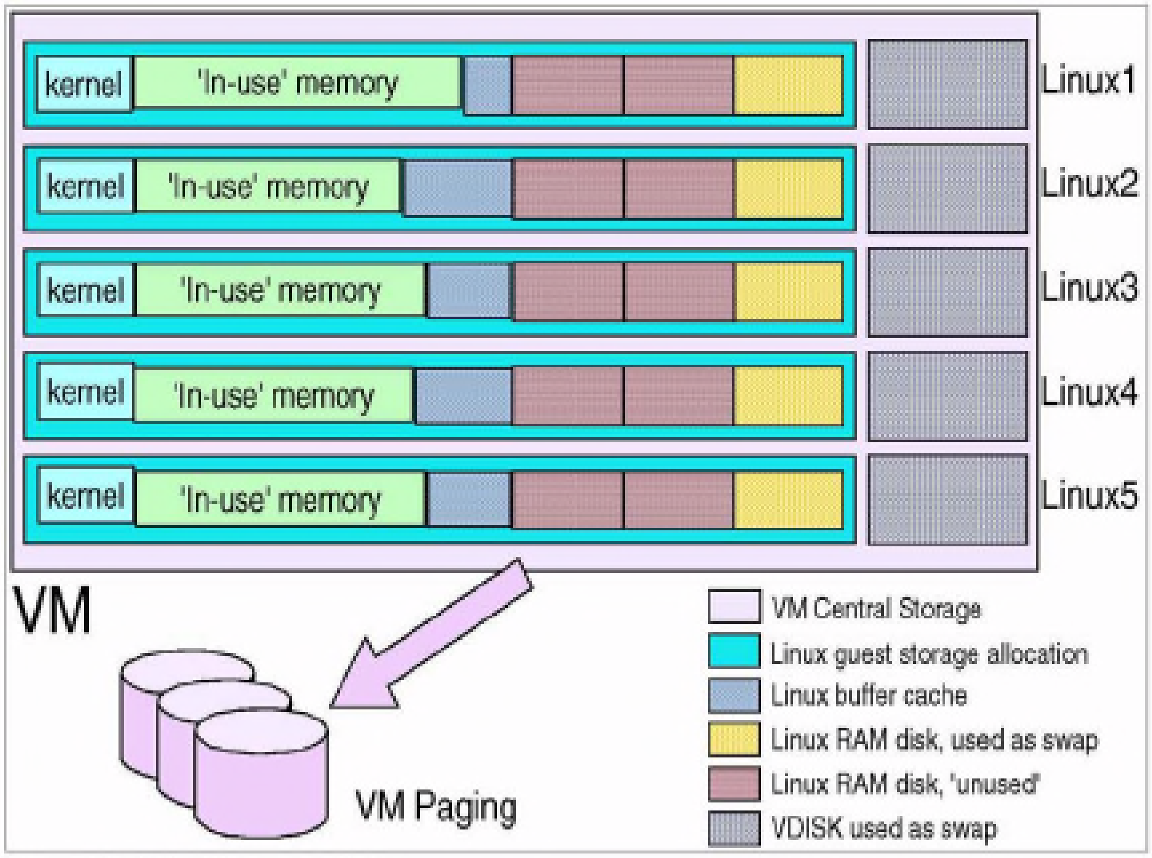


Figure 3-8 Hybrid Linux swap design

As the memory requirement of the Linux image grows, it will start swapping. Initially it will swap to the RAMdisk swap area, but if memory requirements continue to increase, it will fill the RAMdisk and start swapping to the VDISK. At this point, you delete one of the non-swap RAMdisks, releasing that memory back to core.

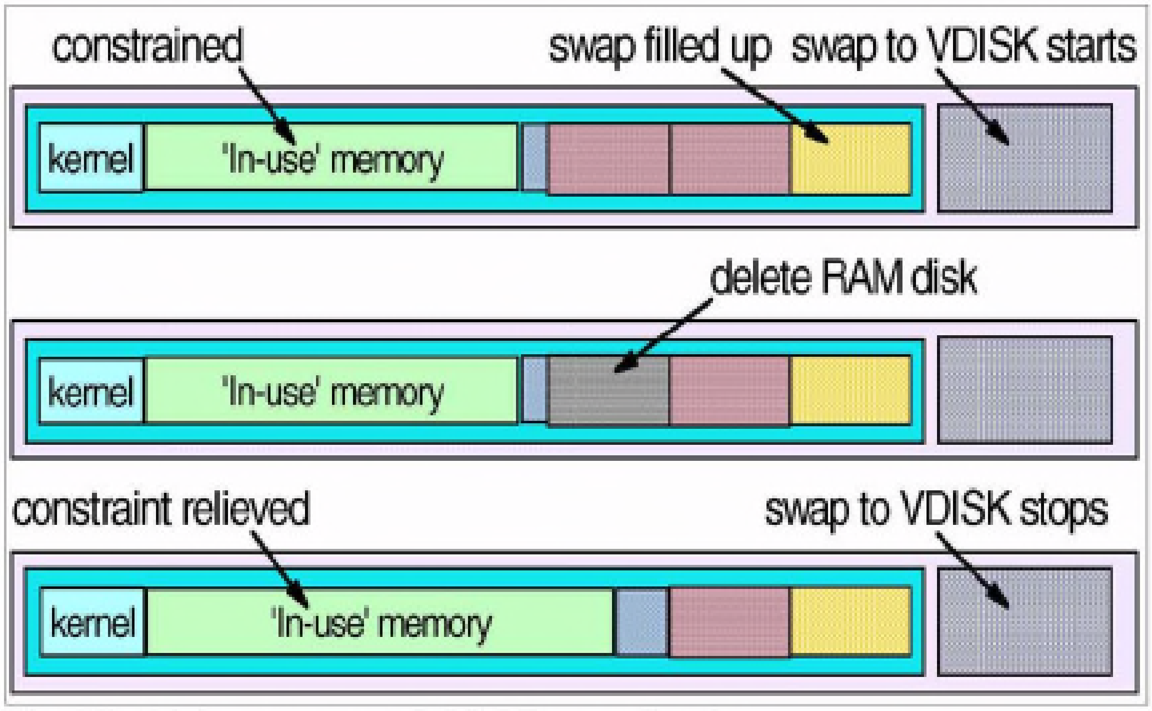


Figure 3-9 Relieving memory constraint in hybrid swap configuration

The process of deleting one of the filler swap disks can be automated. This automation could take place either in Linux (through monitoring of page I/O to the VDISK swap area), or through VM (by monitoring I/O to the VDISK itself). The memory that is used by the files held in RAMdisk will be paged out by VM, so there will be a slight overhead in paging space.

Possibly the best use of this approach is to determine the working set size of Linux virtual machines. By using this approach over time, the ideal configuration for your virtual machine size can be determined.

The process can be further enhanced to provide greater levels of granularity, by adding intelligence that recreates the filler disk and incrementally increases the size of its contents. This can give us a very good understanding of the memory requirement of the Linux instance. (However, there is a risk that a peak in memory utilization would skew the results obtained in this method.)

3.3.6 Sharing Linux memory

In a large penguin colony, many instances of common code such as the kernel will be in memory. VM can provide a shared, read-only storage area that can be used by many instances simultaneously, using a feature known as Named Shared Storage (NSS). This can greatly reduce the memory used by the penguin colony overall.

This shared area can also be used to boot a uniform build of Linux across a large number of instances. Refer to 10.7, "Linux IPL from NSS" on page 228 for more details.

3.4 Network topology

In this section we introduce methods for providing network connectivity to Linux instances in a penguin colony. We look at the connection types available to Linux on zSeries, and the use of these connection methods in connecting Linux to the network. We discuss further details of how the connection types are used in Chapter 4, "Networking a penguin colony".

3.4.1 Network devices for Linux

There are several connection devices available under Linux. Generally, they fall into two categories, external interfaces and internal interfaces.

External interfaces

External interfaces allow Linux instances to communicate outside the boundary of the S/390 or zSeries machine the instance runs in. These are connections to network devices such as Ethernet switches and routers. The interfaces that fall into this category are the Open Systems Adapter (OSA) and Common Link Access to Workstation (CLAW) via ESCON.

Internal interfaces

As the name suggests, internal interfaces are within our S/390 or VM system, and are used mainly to connect between members of our penguin colony. Internal interfaces at this time are point-to-point connections which do not support broadcast. The two internal device types are Virtual Channel-To-Channel (VCTC), and Inter-User Communications Vehicle (IUCV).

Using internal interfaces, members of the penguin colony can communicate with each other, without needing connectivity to a physical LAN. Internal interfaces are the building blocks for providing the virtual networking used in the penguin colony.

In addition to VCTC and IUCV, there are now *HiperSockets* on zSeries machines which were added as this book was being completed. Additionally, z/VM V4R2 allows for virtual HiperSockets and the concept of a Guest LAN. For some details, see 16.2, "VM LAN support" on page 401.

Note: As discussed in "Channel-to-channel (CTC)" on page 74, CTC also operates over physical parallel or ESCON channels, and can be used to link Linux systems which exist in different S/390 processors. Unfortunately, this confuses our classification of CTC a little. The main use of CTC in a penguin colony will be the virtual kind, however, so for the purpose of this discussion, we retain the classification.

3.4.2 Network structure

Since it is not practical to connect all members of the penguin colony to the network using an external interface, it is necessary to create a "virtual" network environment within the mainframe. This will involve having certain parts of the environment set up to do no more than route network traffic. While this takes away some resources from our worker Linux instances, it is a better approach than providing dedicated network access to each individual member of the penguin colony.

Restriction: Be aware that the CTC and IUCV drivers have restrictions on the number of links that can be supported. Refer to "CTC/IUCV device restrictions" on page 102 for more details.

There are many ways that network connectivity can be configured inside the penguin colony. We will cover two of these: a *basic* design providing simple connectivity, and an *advanced* design using multiple interfaces and multipath routing. We then develop the second approach by introducing virtual IP addressing to provide enhanced connection availability.

Simple network design

Figure 3-10 on page 66 shows a basic virtual routing structure.

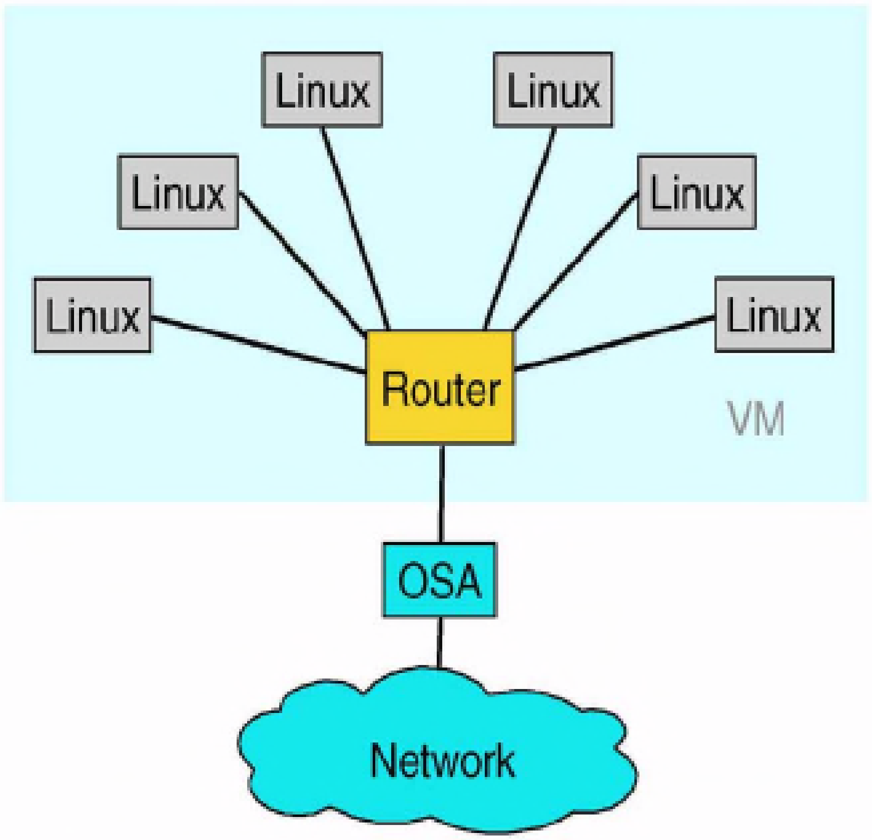


Figure 3-10 Virtual router in a penguin colony

This configuration provides for efficient connectivity, and satisfies the requirement to support many instances using a limited number of network connections.

Resilient network design

For situations requiring a high degree of availability and redundancy, the structure shown in Figure 3-11 on page 67 provides redundant virtual and physical connections.

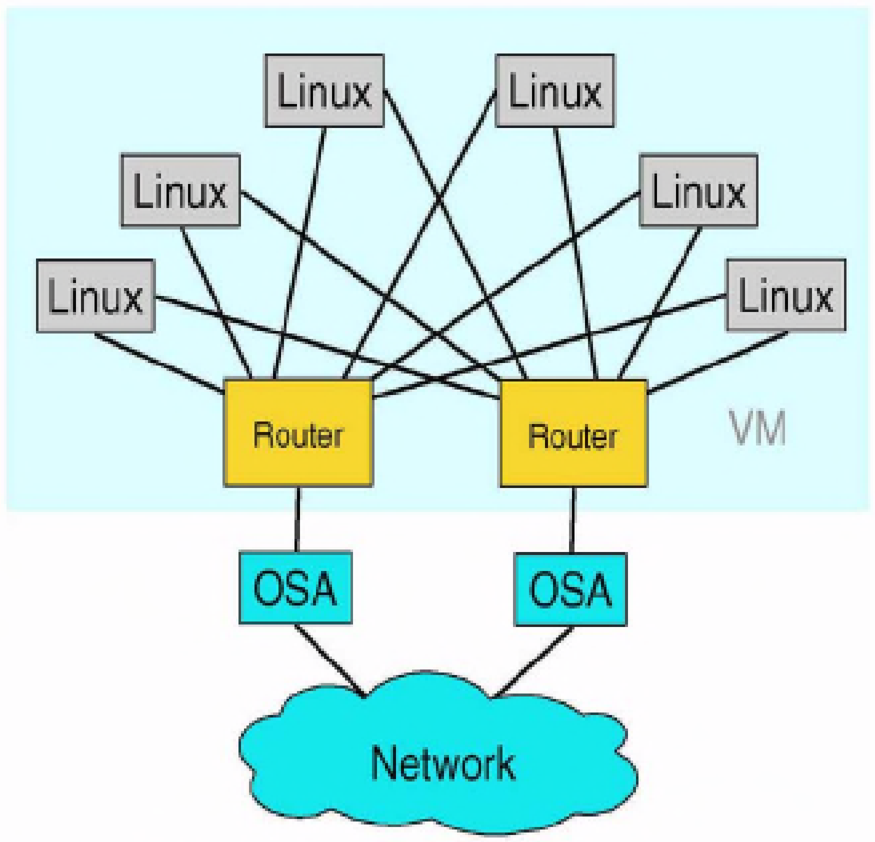


Figure 3-11 Dual router configuration

To properly exploit this design, some kind of dynamic routing is required between the virtual routers and the first line of routers in the network cloud. It may also suit requirements to extend this dynamic routing to include the Linux instances as well.

> **Tip:** Take care in extending dynamic routing too far into the penguin colony, because running **gated** on your Linux instances will result in them having to process and generate routing updates. This will mean that they will wake up (and be awakened) at regular intervals, dramatically increasing the idle overhead of the penguin colony. This situation is discussed further in Chapter 8, "Performance analysis" on page 147.

The role of the router in these examples could be performed by any TCP/IP-capable S/390 or zSeries operating system, depending upon your connectivity method. In this discussion we will consider only Linux and VM TCP/IP.

Linux as a virtual router

The use of Linux in the virtual router role has the following benefits.

- Efficient routing engine
- Linux's advanced routing capabilities, including Netfilter
- Ability to build a thin, "routing only" kernel to maximize performance

When adding new members to the penguin colony, however, the CTC and IUCV drivers must be unloaded in order to add the new connections. This is disruptive to existing connections (and in the case of the CTC driver, may cause other servers to be restarted). In addition, the scalability of the CTC and IUCV drivers is not extensively proven.

The new HiperSockets support, which was announced as this book was being completed, will allow another networking option for zSeries machines. z/VM V4R2 provides support for both physical HiperSockets, which have a prerequisite of z900 hardware, and virtual HiperSockets. As well as running on zSeries machines, virtual HiperSockets will also run on machines that z/VM V4R2 supports: G5, G6 and Multiprise 3000 models.

VM TCP/IP as a virtual router

Using a VM TCP/IP user as the virtual router addresses the availability issue of the Linux router, because new devices can be added on the fly using OBEYFILE.

Note: The TCPIP OBEY command in VM allows the dynamic alteration of a running TCP/IP stack using configuration statements contained in a file known as an OBEYFILE. Refer to VM TCP/IP publications for more details on how this works.

Also, using VM as a "first-level" router can provide a more proven method of providing connectivity to Linux using unsupported or experimental network hardware (for example, Cisco CIP, for which the Linux driver is still considered experimental).

There are still scalability issues, however, as there are hard restrictions in VM as to the number of devices available (see "VM TCP/IP restriction" on page 104). Also, while it is generally considered that the performance of the Linux kernel would be superior to VM TCP/IP in routing IP packets, it is probably not significantly better. As it is necessary to set up at least one VM TCP/IP stack simply to support the VM system itself, the VM TCP/IP stack is a good choice for general routing use.

Combining Linux and VM TCP/IP for routing

In certain high-demand applications, it may be advantageous to combine Linux and VM TCP/IP routing in a layered approach. This may be necessary as a result of limitations in the network devices being used, or because the network interfaces available do not support Linux and you still wish to use Linux routing features such as Netfilter.

Note: Netfilter becomes available in the Linux 2.4 kernel, and provides firewall and traffic monitoring functions. It is the replacement for ipchains from the 2.2 kernel, and provides more functionality, including stateful firewalling.

Figure 3-12 illustrates a possible configuration (note that this is only an example; a real configuration may vary from this).

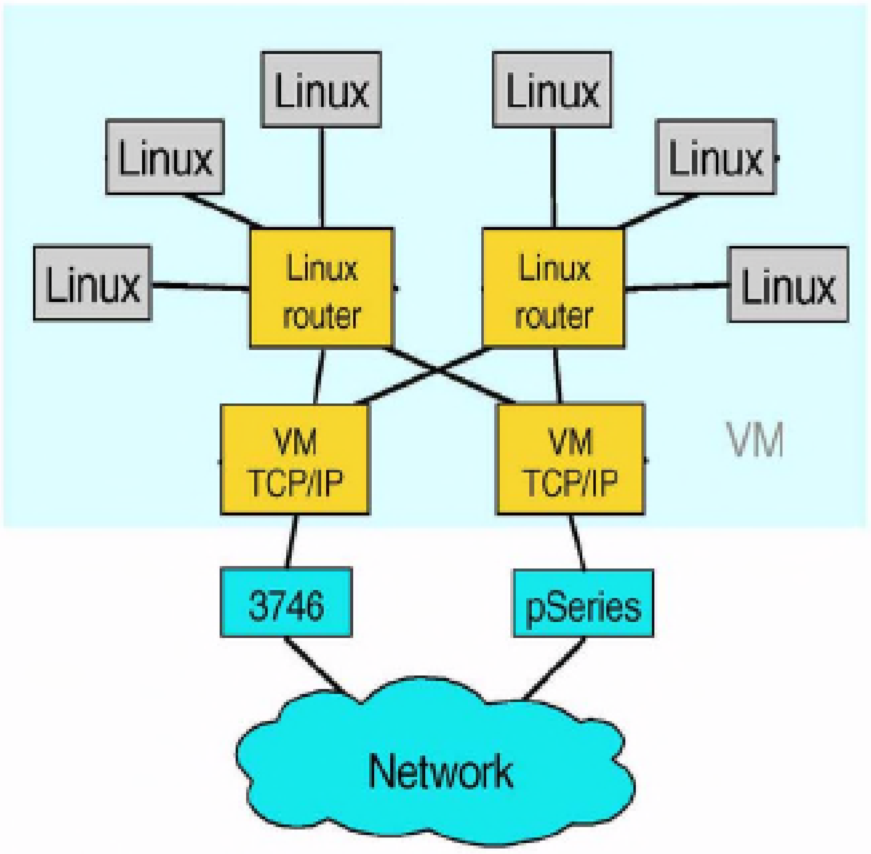


Figure 3-12 Combining VM TCP/IP and Linux virtual routing

Refer to Chapter 4, "Networking a penguin colony" on page 73, for further discussion about hierarchical routing structures.

3.4.3 General network considerations

In this section, we discuss general considerations you should keep in mind regarding the design of the network.

Talk to your friendly network staff

The success of this kind of design is dependent upon good communication and discussion with the network design team at your installation. While we've drawn "The Network" in our figures as the ubiquitous cloud, there is a lot of complexity in that cloud that must be considered by the penguin colony designers, just as there is complexity in the penguin colony that must be considered by the network team.

One of the most difficult concepts for many network staff to grasp is that we are building entire routing domains inside a single physical host. We design links, routers, and hosts that exist virtually inside VM—but in spite of their being virtual, they behave the same as physical links, routers, and hosts.

Tip: You may have noticed that the figures in this chapter have the surrounding VM system shown in very light shading. This is done for a reason, and we recommend you do the same with your own diagrams.

We suggest that, when presenting your designs to the network staff, you photocopy the diagrams with a light setting on the photocopier so that the box representing VM does not show up. In this way, the "virtual" network appears just like a network with real devices. At a later time, show the real diagrams which include VM. You can use this method to introduce people to the concept of virtual networking in a penguin colony.

Spend some time with your network team, discussing the interaction between the network and the penguin colony. Issues including physical connectivity and dynamic routing will have to be clarified as part of the design of the entire solution. Refer to Chapter 4, "Networking a penguin colony" on page 73 to find more about the issues that you will need to cover.

3.5 Workload tuning

In a penguin colony of any size, it is important to remember that all of the Linux instances are sharing the resources of the S/390 processor complex. In practice, this means that processor capacity used by unnecessary tasks should be minimized. Some tips would include the following.

- Run only the services you need to do the work you want. This ensures that the server is working at one task only. (You will see this recommendation from a security perspective, also.)
- Minimize the number of services that "poll", or at least be aware of their impact. Examples of this include `nmbd`, the Samba NetBIOS name server daemon, and `gated`, the dynamic routing daemon.
- Where possible and convenient, use a service to back itself up rather than running additional services. In this way, the overhead of the server is reduced by not having to provide a backup service specifically. This is discussed further in "In-service backup" on page 129.

4

Networking a penguin colony

This chapter expands some of the issues introduced in Chapter 3, “Virtual server architecture”, related to connecting your penguin colony to your network.

4.1 Network devices

Networking on Linux for zSeries and S/390 offers many physical and virtual options.

4.1.1 Open Systems Adapter (OSA)

The OSA is a network card for S/390 and zSeries mainframes. Linux can utilize Ethernet, Fast Ethernet, Gigabit Ethernet and Token Ring OSAs. The OSA-Express is the newest version of the OSA, providing a number of improvements and new features, as follows:

- New Gigabit Ethernet interface, and improved Fast Ethernet interface
- Queued Direct I/O (QDIO) mode, which uses the Self-Timed Interconnect (STI) bus to provide memory-to-memory transfer of network traffic, dramatically increasing throughput
- IP Assist feature (available in QDIO mode), which offloads adapter processing from the host TCP/IP stack
- Dynamic OSA Address Table (also in QDIO mode only), which eliminates the need to predefine IP addresses
- Increased number of IP addresses per port available in OAT: from 16 to 512 on G5/G6 to 2048 on zSeries, and a maximum of 240 entries per OAT
- Increased number of subchannels available
- LCS support for TCP/IP when configured in non-QDIO mode

OSA provides a direct path to the "outside world" where required, but not all members of a large penguin colony can connect to it. To provide network connectivity to the entire penguin colony, virtual networking is required. These issues are discussed in 3.4.2, "Network structure" on page 65.

4.1.2 Channel-to-channel (CTC)

CTC is a point-to-point connection, using either a real S/390 channel interface or a virtual channel provided by VM. Multiple CTCs can be configured, allowing connectivity to multiple points in the environment.

The link-layer protocol used by S/390 systems for TCP/IP over CTC is very similar to Serial Line Interface Protocol (SLIP), an early TCP/IP connection method used over dial-up communications lines. Because all S/390 operating systems use the same link protocol, it is possible to connect a Linux instance not only to another Linux, but also to a VM or OS/390 TCP/IP stack.

VM Virtual CTC (VCTC) allows a device pair to be configured that links devices on two guests together. These linked devices then act exactly like a physical CTC, with VM carrying the network traffic internally between the two guests. Because no physical I/O is performed, VCTC often performs far better than a physical CTC. However, VCTC can only be used between guests in the same VM system (that is, a VCTC cannot link two guests in different VM systems, even on the same physical machine).

CTC support is provided in Linux using code contributed to the kernel tree by IBM. The CTC driver can be compiled as part of the kernel, or a module (ctc.o). CTC has shown high reliability in operation under Linux. However, it cannot be dynamically configured (you cannot add a new CTC device without unloading and reloading the CTC module, thus terminating any existing CTC connections). Also, the CTC driver becomes unstable if the channel is interrupted for any reason (that is, if the other end of the link is brought down).

Note: The latest versions of the IBM kernel patches include support for emulation of a raw serial connection over CTC. It can be used to provide a Linux-to-Linux connection without TCP/IP, which could be used for a console device.

4.1.3 Inter-User Communications Vehicle (IUCV)

IUCV provides a virtual communication link which is similar in function to VCTC, but does not involve channel I/O. It is available only under VM, and can be used only to connect between Linux images or VM TCP/IP stacks in a single VM system. Some prior planning is required to ensure that guests have the right directory authorization to make IUCV connections.

Like CTC support, IUCV support is IBM code that has been contributed to the kernel, and can be compiled either monolithically or as a module. From the Linux configuration perspective, it is still a point-to-point TCP/IP connection, so configuration issues are almost identical to CTC.

Because IUCV does not have to emulate channel I/O, performance is better than VCTC. While the driver has the same lack of dynamic configuration as the CTC driver, it has shown more resiliency against interruption of the communications path than CTC.

4.1.4 Other devices

In the following sections, we discuss other networking options.

Common Link Access to Workstation (CLAW)

This device type was originally used to provide TCP/IP connectivity to channel-attached RS/6000 machines, but is now also used to connect to the Cisco Channel Interface Processor (CIP) card. This would allow installations using channel- or ESCON-attached Cisco routers to link Linux systems directly, in order to decrease the routing path between the network and the Linux instance.

An experimental driver for CLAW has been written by UTS Global, but has not been extensively proven. An alternative to using this driver would be to use TCP/IP on VM to link to the hardware, then use IUCV to communicate between Linux and VM (this is illustrated in "VM TCP/IP as a virtual router" on page 68).

Other S/390 TCP/IP devices

There are a number of other connection types supported by S/390 hardware, including:

- Channel Data Link Control (CDLC), used to connect to the IBM 3746 Multiprotocol Controller
- Multi Path Channel (MPC), enhanced protocol used for OSA (non-QDIO) and host-host links
- ATM and FDDI OSAs
- HYPERchannel A220 devices
- SNALINK (TCP/IP access over SNA to 3745)

While Linux does not natively support these connections, it is possible to use a z/VM or OS/390 system to connect to them, and then use CTC or IUCV to link to the Linux system.

In some cases it may be possible to reconfigure the hardware to provide direct connectivity to Linux. For example, if you are using the MPCOSA device type for communication with an OSA in OS/390, you could create an LCS definition for use by Linux.

4.1.5 HiperSockets

HiperSockets were annouced after this book was nearly completed, so they are only mentioned. They were developed to provide a highly available, high-speed network connection among mainframe operating systems, such as z/OS V1R2 or later, z/VM V4R2 or later, and Linux distributions with such support. This integrated any-to-any TCP/IP network provides a list of benefits not achievable by grouping servers around a z900 interconnected by external networking technology. See the following Web site for details:

http://www-1.ibm.com/servers/eserver/zseries/networking/hipersockets.html

4.2 Resilient IP addressing

In 3.4.2, "Network structure" on page 65, we show ways to create reliable physical connectivity for the members of our penguin colony. However, more work has to be done to provide resilient connectivity to applications.

Analogy: A person who wants high availability telephone service might have a second phone line installed. However, without listing both phone numbers in the telephone book, or installing an automatic diversion from one line to the other, the second line would be useless since no one would know to dial it.

In the same way, multiple IP addresses have to be tied to a single external iterface in Linux.

Many methods are available to do this, including:

- DNS manipulation (round-robin, dynamic)
- Connection balancing
- Virtual IP addressing

4.2.1 DNS manipulation

Round-robin

The DNS can be altered to resolve name requests to multiple IP addresses. This is known as round-robin DNS. The DNS zone file lists multiple DNS A records for the one host name, and DNS will issue each address in sequence in response to requests for that name. The following is an example:

```
vmlinuxa  IN A  9.12.6.69
          IN A  9.12.6.67
```

The A records can be specified in different orders, or multiple times, in order to weight the different adapters and bias traffic towards or away from certain adapters. For example, in the preceding example, if the first IP address was associated with a Gigabit Ethernet interface and the second was a Fast Ethernet interface, more connections could be directed to the Gigabit Ethernet interface by listing that address more often than the other, as follows:

```
vmlinuxa  IN A  9.12.6.69
          IN A  9.12.6.69
          IN A  9.12.6.69
          IN A  9.12.6.69
IN A  9.12.6.67
```

This would result in the address of the Gigabit Ethernet adapter being given out four times more often than the Fast Ethernet interface.

Note that this is *not* a solution for high availability, since in this model the DNS server cannot respond to the failure of an interface, and will continue to serve the address of a failed interface in response to requests. In the example, if the interface at 9.12.6.69 failed, four out of five connection requests to vmlinuxa would also fail.

Another issue to consider is DNS caching at the client. In order to balance connections properly, these installations rely on specifying a short Time-To-Live (TTL) value in the returned DNS record. This is intended to stop clients from caching the DNS information, so that repeated requests from clients are distributed across all servers. It has been found, however, that some client applications and platforms ignore the TTL value returned from the DNS server. This might cause a particular client to continually request service from a non-preferred interface simply because its first request happened to return that address.

Dynamic DNS

Making the DNS server respond to the state of interfaces and applications is done using dynamic DNS. The DNS zone is updated by adding and deleting records on the fly according to factors including load, available applications and interface state.

Many solutions are possible using dynamic DNS, and it can also be a prerequisite to some designs based on virtual IP addressing (see "Virtual IP addressing" on page 79). One example would be to update the DNS to add an address record when a new interface is added or removed. It is also possible to integrate with network management utilities to add or remove DNS records relative to the amount of traffic being carried by the available interfaces.

Like the round-robin DNS option, dynamic DNS is subject to caching of DNS records in the client. This means that a client may have an extended outage due to caching of DNS data, even though the DNS server has been correctly updated to reflect the outage of an application server. Also, dynamic DNS is not yet standardized across DNS server implementations. For example, the Berkeley BIND-based DNS server generally distributed with Linux has implemented a method for dynamic updating which differs from the method used by the VM TCP/IP DNS server. While these differences can be designed around, it adds to the complexity of the design.

4.2.2 Connection balancing

In a connection-balancing scenario, an external (to the Linux hosts) service fields incoming connection requests and serves them to the real servers. This approach is used in clustering solutions, and is beyond what is required to provide network redundancy for a single host. However, a simplified version of this method could be used at a virtual router to share traffic across interfaces even on a single machine.

One of the great advantages of this approach is that it eliminates the problems associated with caching of connection and DNS information, since the IP address of the distribution server is used for all requests. Unfortunately, this same feature is the greatest downfall, because in a simplistic design the IP address of the distribution server becomes a single point-of-failure.

For more information on this kind of design, refer to Chapter 18 of *Linux for zSeries and S/390: Distributions, SG24-6264*.

4.2.3 Virtual IP addressing

z/OS and z/VM provide a facility called Virtual IP Addressing (VIPA), which creates an address within an IP stack that can be used to isolate applications from the failure of an IP interface. Applications bind to the virtual address instead of an interface address. If a physical interface fails, dynamic routing protocols can reroute traffic over other active interfaces, without operator intervention or outage.

Linux provides a dummy network interface that can be used to provide a similar feature. The dummy interface has an IP address, but is virtually associated with the physical interfaces of the Linux host.

Support for the dummy interface must be added to the kernel, either as a module or monolithically. You may need to add the dummy device support to your kernel before you can use it (our SuSE distribution did not have the driver available, so we had to build it).

Tip: If you have a kernel configuration file available, you can issue the following command to check the configuration status of the dummy device:

```
cat <config_file_name> | grep DUMMY
```

Remember that on the SuSE distribution, the pseudo-file /proc/config.gz contains the configuration of the current kernel. Copy this pseudo-file to another location and uncompress it to obtain the kernel configuration file.

A dummy interface can be added to the resilient design shown in Figure 4-1 on page 80, producing a design that allows almost transparent backup over the failure of any one of the network connections. This is shown with respect to a single Linux image in the figure.

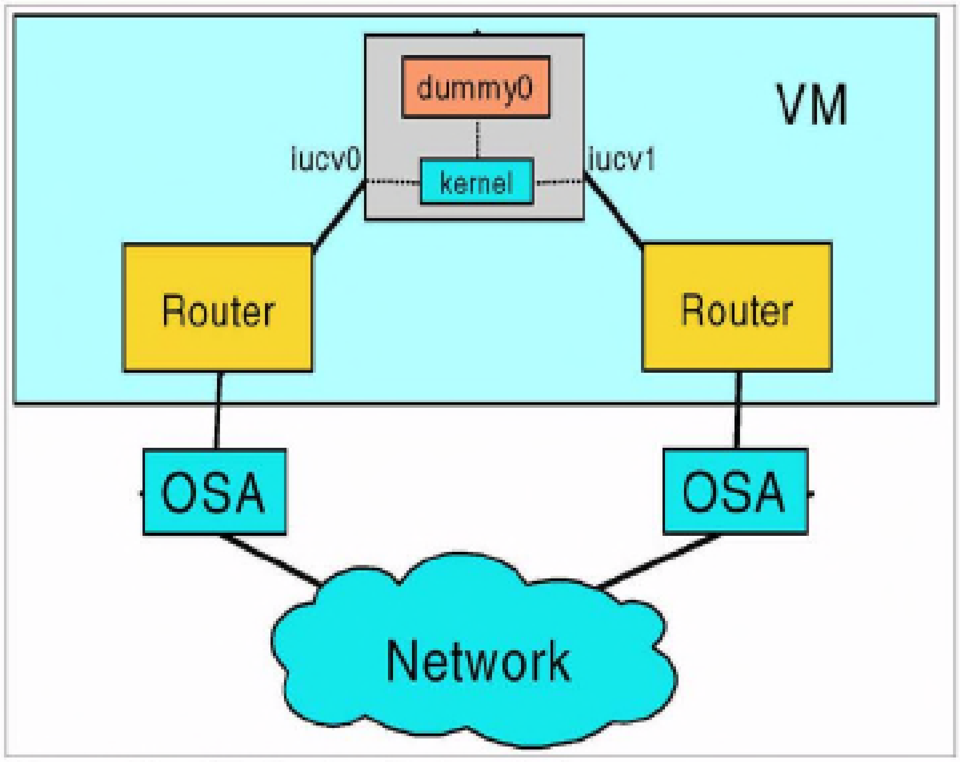


Figure 4-1 Virtual IP addressing using dummy interface

The IP address associated with the Linux image in DNS and other facilities is the IP address configured on the dummy0 interface. Using IP routing, incoming traffic is sent to this address using either network interface. If one interface fails, IP routing can forward the traffic via the other interface.

While this works for incoming traffic, any outbound traffic the Linux instance generates is associated with one or the other physical interface. In z/OS and z/VM, the SOURCEVIPA configuration support provides a means to source IP packets from the virtual address, removing the dependency on the physical device. When using SOURCEVIPA, traffic originates from a virtual interface rather than a real interface.

This support is not available under Linux at this time, so TCP/IP sessions that originate from a Linux instance using the dummy driver will fail if the interface those sessions originate on fails. Refer to 17.2, "SOURCEVIPA equivalence for Linux" on page 409 for a discussion about how this might be provided under Linux.

4.3 Packet filtering and Network Address Translation

Whenever you connect your computer to today's Internet world, you are exposed to intruders from the outside. There are thousands of hackers just waiting to get into your computer to do damage or, perhaps, steal information. You need protection against them.

As already mentioned for general networking, our penguin colony is no different from separate servers. Therefore, our penguins need to be protected also. There are additional complications in a penguin colony, however, because you are likely to have many different networks—possibly for different customers—all sharing the same VM system. Not only does the penguin colony need to be protected from outside attack, but members of the penguin colony must be protected from each other as well.

This section introduces the IPTables utility, which controls the Netfilter features of the Linux 2.4 kernel. Netfilter provides very strong traffic filtering and translation capabilities, and is suitable for even some advanced firewalling roles inside your penguin colony.

Important: Although we provide an introduction to IPTables and Netfilter, and instructions on how to set up a capable firewall using them, we are not security experts. These chapters are not substitutes for a proper security assessment and implementation. Depending on your security requirements, you may still require dedicated firewall equipment outside your penguin colony.

4.3.1 What is packet filtering

As you can tell from the name, packet filtering is a method of filtering data coming to your computer, and is commonly used in firewall implementations.

Because everybody wants to communicate, sooner or later you need to connect your private network to the Internet; this has security considerations. With packet filtering, you can implement a firewall that will protect your computer from the outside world. You can also use a firewall on a single computer which, for example, is connected to the Internet through a dial-up line.

When you install a firewall to protect your internal network, every external computer that wants to talk to a computer on the internal network must ask the firewall for permission. If the permission is not granted, access is denied.

4.3.2 What you can do with Linux packet filtering

Using Linux packet filtering, you can do the following:

- Protect your internal network, that is connected to the Internet, from outside intruders
- Perform Network Address Translation (NAT), which allows internally-connected computers without a registered Internet address to reach Internet resources
- Filter the information going in or out of your internal network (or in or out of just one computer)
- Use your Linux server as a gateway between two different types of networks, for example, connecting Token-Ring and Ethernet worlds; this can be a cheap solution in comparison to buying an expensive router
- Share your dial-up Internet connection with others

4.3.3 Planning for packet filtering implementations

In this section, we show some of the possible designs for networking infrastructures used for packet filtering implementations. In our test environment, we always assume that the OSA card is configured to pass *all* traffic to a VM guest, not just the predefined addresses.

In the OSA card, only one IP address can be defined as the primary route (using OSA/SF), which means that this address will pass through *all* packets—not just the packet with the destination address for this IP address. When the OSA card is defined to have the primary route, we always use a VM Linux guest as the entry point for all traffic.

In the following examples, we show a few different approaches for setting up an internal network for your VM Linux guests.

Single router/firewall with one server

In the scenario shown in Figure 4-2, we have a single Linux image which acts as a router and firewall for the Linux server on an internal network. In this case we have only one server.

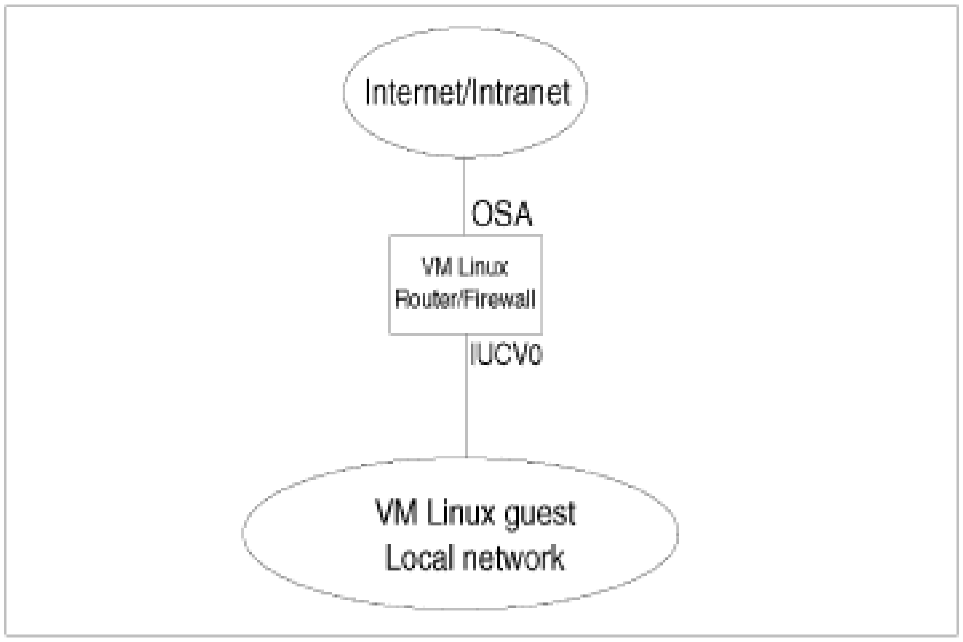


Figure 4-2 Single router/firewall implementation

Single router/firewall with DMZ on one server

In the scenario shown in Figure 4-3, we use a DeMilitarized Zone (DMZ) to host our Web and mail server. Because the DMZ is hosted on a separate interface, the other server can be protected from external traffic.

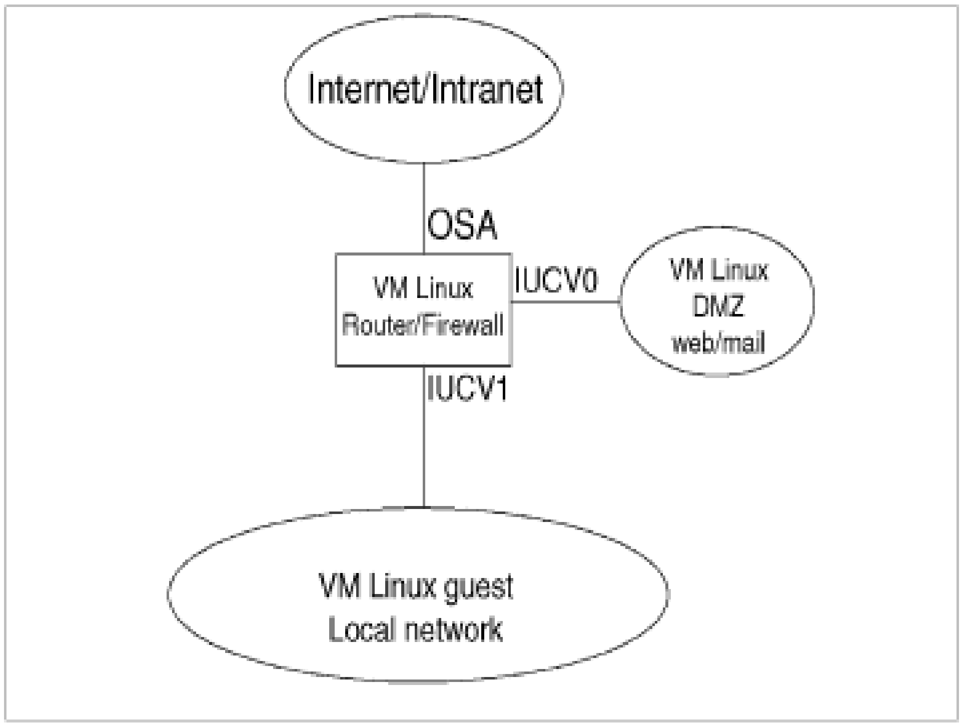


Figure 4-3 Single router/firewall with DMZ

Single router/firewall with more servers

In Figure 4-4, we use a single Linux image as a router/firewall for more servers.

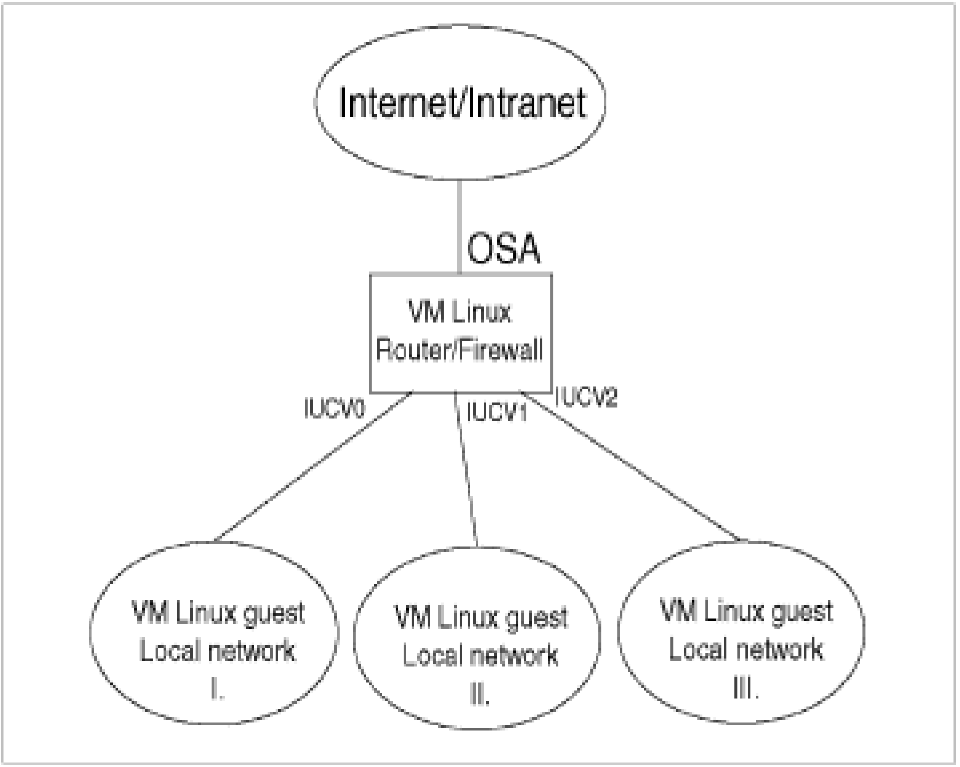

Internet/Intranet
OSA
VM Linux Router/Firewall
IUCV0
IUCV1
IUCV2
VM Linux guest Local network I.
VM Linux guest Local network II.
VM Linux guest Local network III.

Figure 4-4 Single router/firewall with more subnets

In this scenario, each server is connected to the router/firewall via an independent interface. This means that we can control packet routing in such a way that only the correct packets are being forwarded to the servers. For example, we can block all broadcast traffic to our servers.

This kind of solution is often used when all servers belong to the same owner. In this scenario, we could also implement the DMZ for services that must be separated from the others.

Two-layer router/firewall implementation

In the scenario shown in Figure 4-5, we use two layers of routers/firewalls. The first-layer router/firewall is for the up-front Internet connection that serves the second layer of routers/firewalls. The second-layer firewalls are used to isolate each local network from the others.

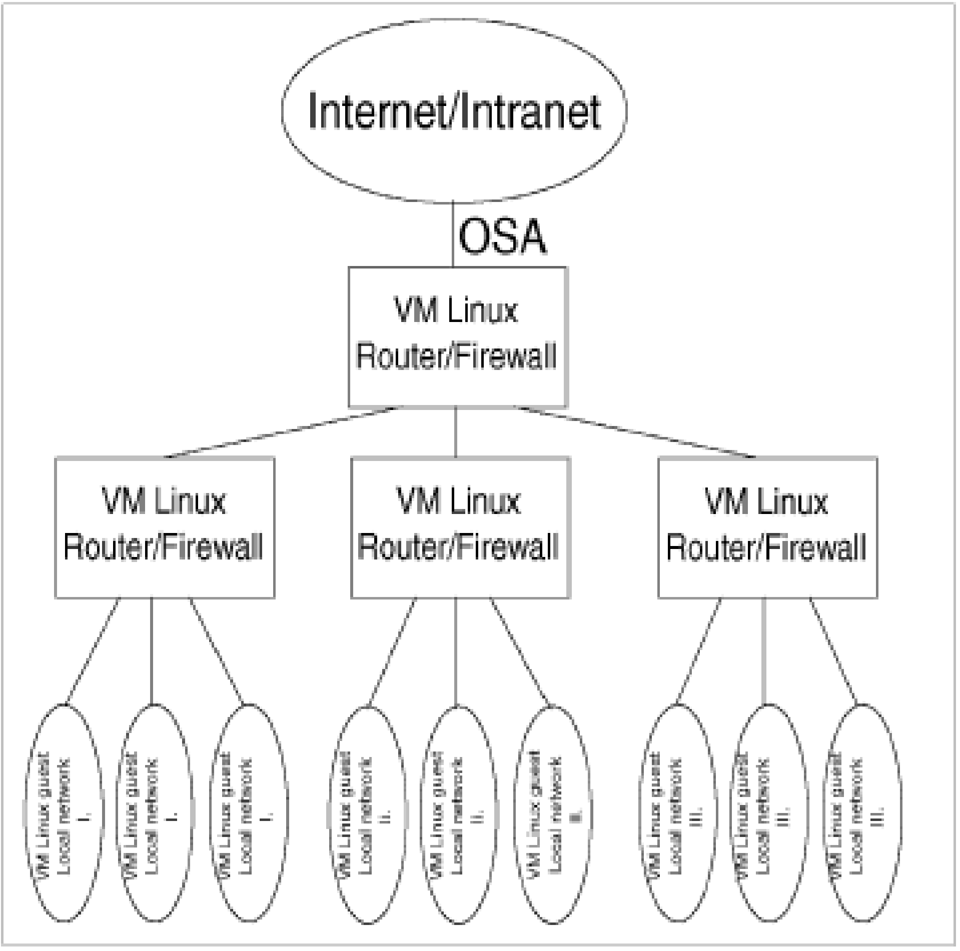

Internet/Intranet
OSA
VM Linux Router/Firewall
VM Linux Router/Firewall
VM Linux Router/Firewall
VM Linux Router/Firewall
VM Linux guest Local network I.
VM Linux guest Local network I.
VM Linux guest Local network I.
VM Linux guest Local network II.
VM Linux guest Local network II.
VM Linux guest Local network II.
VM Linux guest Local network III.
VM Linux guest Local network III.
VM Linux guest Local network III.

Figure 4-5 Two-layer router/firewall implementation

With such a solution, you provide an independent network for each small colony of Linux servers that's connected to the outside world over its dedicated router/firewall. Also, none of the servers in the local network interferes with the others, because it is connected to the router on its own interface.

This approach is also needed to implement a big colony of servers in the single-box solution. On the first router/firewall, we accept all traffic from the Internet. By implementing packet filtering, we can get rid of unnecessary packets (i.e., broadcasts) and pass only packets that are going to the services on our servers.

On the second layer, we separate servers among different owners for security reasons. On this layer, we also take care that the packet traveling to a particular server does not hit the interface of other servers in the local network. With this implementation, we ensure that we do not wake up any idle servers without good reason.

This approach also simplifies the management of the Linux colony inside the box, because the connections have to be predefined on VM and also in Linux. By separating each server in small local networks with its own router/firewall, you have a manageable number of servers (for example, defining connections for ten servers in the router is much simpler than defining connections for hundreds of them).

4.3.4 How packets travel through a gateway

In this section we will explain how IP Tables work. You can see the path of a packet coming into your server in Figure 4-6.

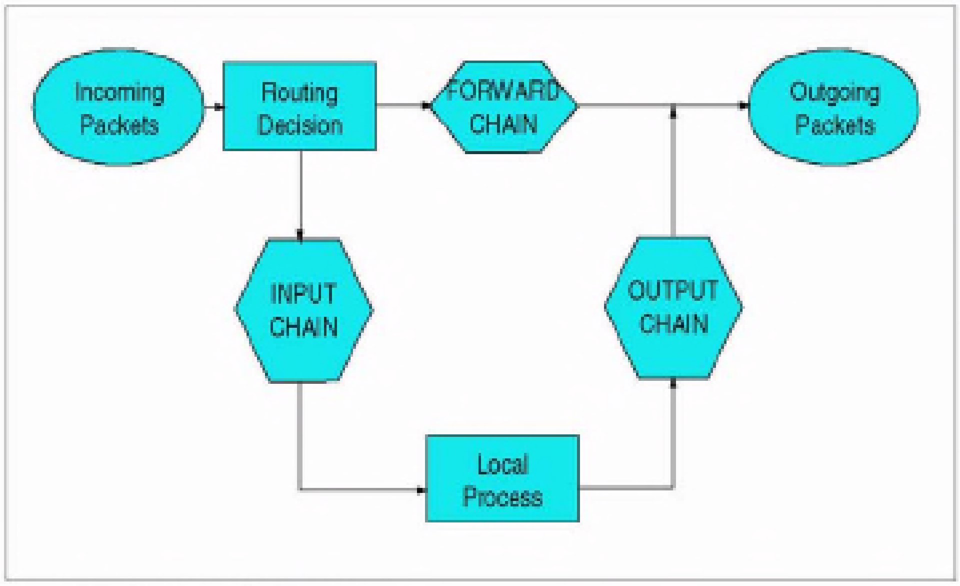


Figure 4-6 How a packet travels

The following list gives brief descriptions of each stage:

Incoming packets	This is where the packet enters the gateway.
Routing decision	At this point, the kernel decides whether the packet will be forwarded to another interface, or if it will be sent to a local process.
Input chain	Before the packet gets to the local process, the kernel checks the rules in this chain against the packet.

Local process	These are applications or processes running on the server.
Output chain	Before a packet from the local process is sent to the output interface, the kernel checks the rules in this chain against the packet.
Forward chain	Packets that only travel through the router from one interface to another are checked against the rules in this chain.
Outgoing packets	This is where the packet exits the server.

As you can see from Figure 4-6 on page 87, there are three places where you can check the packets in your server:

- Input chain
- Forward chain
- Output chain

In 11.2.6, "Using IP Tables" on page 256, we explain how to use basic IPTables rules to filter traffic in your Linux instance.

4.3.5 Network Address Translation (NAT)

Network Address Translation (NAT) is used to translate the source or destination address of the IP packets traveling through the gateway. If you are using a Linux router/firewall to provide connection between the Internet and a local LAN, you can use NAT to conserve IP addresses. The router/firewall uses a real IP address, and to avoid wasting real IP addresses for the machines on the LAN, NAT is used on the router/firewall to allow the machines on the LAN to use *private IP addresses* (10.x.x.x or 192.168.x.x) to access the Internet transparently. The users see no difference between private and real IP addresses.

When packets are leaving the LAN, the router/firewall uses NAT to replace the source address of the originating computer with its source address. The destination computer replies back to the router/firewall, and the destination address is modified to send it to the computer on a private LAN.

There are three basic types of NAT:

1. SNAT (Source NAT) - in this process, the source address of the first packet is modified. SNAT is always done *after* the routing, just before a packet leaves the router/firewall.
2. DNAT (Destination NAT) - in this process, the destination address of the first packet is modified. DNAT is always done *before* routing, just after a packet enters the router/firewall.

3. Masquerading - this is a special form of SNAT.

We discuss each type in greater detail in the following sections.

Source NAT (SNAT)

With SNAT, you change the source address of the packet. This is done before packets are sent out. All other processes on the router (routing, packet filtering) will see the packet unchanged.

SNAT is used to change the address that a packet appears to come from. This is useful in applications like server clustering, where the IP traffic must appear to originate from the cluster IP address and not from the individual server that handled the request.

Destination NAT (DNAT)

Destination NAT involves altering the destination address of the packet. This can be used in Web proxy systems, where a firewall or router transparently redirects a request from a client to a cache system on your local network.

DNAT is done just after the packet comes in. All other processes (routing, packet filtering) will see the packet with the destination as you have altered it. This is important, because if the packet is routed prior to DNAT, it might be sent to the wrong interface for the altered destination address.

Masquerading

This is the specialized version of SNAT. It is usually used for dynamically assigned IP addresses (for example, dialups). With masquerading, you do not need to put in the source address; instead, the source address of the interface where the packet is going out will be used.

See 11.2.11, "Using IPTables for NAT" on page 263 for a description of how to use NAT and how to set it up using the iptables tool.

4.4 General network considerations

In "Talk to your friendly network staff" on page 70, we introduced the idea of discussing connectivity issues with the network team at your installation. This section introduces some of the issues that need to be addressed during these discussions.

4.4.1 How penguin herding is different

For newcomers to the concept of running Linux guests under VM, the idea of virtual networking connections between guests can be a bit difficult to grasp. While it is possible to give a number of your Linux instances access to the network directly, this would not be making efficient use of the hardware and the architecture. Through the use of *virtual* networking, a large amount of the network infrastructure traditionally present in a server farm can be eliminated.

Important: We do not suggest that *all* network tasks be brought into the penguin colony. Tasks such as high-throughput firewalling and management of modem banks do not benefit from being moved to S/390. However, savings can be realized by, for example, not having to provide a network port for each and every Linux instance.

In some ways, connectivity inside a penguin colony is somewhat easier than in a discrete server environment. This is because network connectivity is over non-broadcast point-to-point links, which helps make the environment more secure than using physical LANs. The routing infrastructure can become quite complex, however, because of the number of point-to-point links that must be managed.

Note: Refer to 13.7.1, “NetSaint” on page 323, for a discussion of the use of this open source tool, which we found useful in diagramming and graphically managing the network connectivity of a penguin colony.

When designing virtual routing systems for a penguin colony, remember to treat virtual routers the same way as real routers when it comes to designing your routing domains.

Virtual networking can be viewed from a network perspective, and from a processing perspective.

Virtual networking from a network perspective

In your virtual routing environment, the devices and links between virtual servers will be functionally identical to real ones. Virtual network devices behave just like real devices do, and the routing done in virtual routers is no different from routing in a physical machine.

In addition, because many basic packet-filtering tasks can be performed in your Linux routers, you can avoid the need for a lot of external firewall equipment except for high-security applications (as mentioned previously).

This is why, when designing the network connectivity, the choices to be made are no different from the choices that exist in the physical world.

Virtual networking from a processing perspective

In the world of physical routers and networks, the server people need give no consideration to the CPU consumed by routers and network equipment. The fact that a router actually contains a CPU and performs work is overlooked by everyone except the network people who support them. And even network people will disregard the fact that a modem also has processing function and requires CPU (or perhaps some analog equivalent).

In your penguin colony, however, routers and links are virtualized by VM, and share the CPU consumption of the penguin colony. This means that virtual network traffic will cost real cycles in your S/390 processor. Also, every virtual interface defined will require buffer space, and this will add up over all the machines in your penguin colony.

So this is perhaps the most important thing to remember in terms of networking your penguin colony: in return for what you save in terms of physical network equipment (routers, switches, network ports), you have to give up some S/390 processor and memory capacity to run your virtual network.

4.4.2 Dynamic routing

Almost all large TCP/IP networks today use some kind of dynamic routing to maintain connectivity. In our penguin colony, we must interact with the rest of the network and participate in the routing decisions that occur.

Dynamic routing involves routers exchanging information about the state of the network. Several protocols are used to provide dynamic routing, including the following:

- Routing Information Protocol (RIP)
  This early dynamic routing protocol uses distance-vector calculations to build a view of the router network.
- Open Shortest Path First (OSPF)
  This link-state protocol uses the status of the links between routers to calculate the shortest path between endpoints.
- Enhanced Interior Gateway Routing Protocol (EIRGP)
  Developed by Cisco Systems, this protocol uses a proprietary algorithm which combines benefits of distance-vector and link-state algorithms.
- Border Gateway Protocol (BGP)
  This routing protocol is used to provide reachability information between

separate networks, allowing connectivity between them without exchanging topology information.

Interior gateway protocols are usually used within an organization to provide high levels of connectivity between various parts of the internal network. RIP, OSPF and EIGRP are all interior gateway protocols. *Exterior gateway protocols* do not exchange topology information, and as such are used to connect networks belonging to different organizations (such as a company and their ISP). BGP is an exterior routing protocol.

Tip: For more information on dynamic routing protocols and their implementation, refer to the abundant documentation that available on this topic.

The exact method used to "Internetwork" your penguin colony with the rest of your corporate and/or customer network will vary, depending on a number of issues, including:

- The routing protocol currently in use in your network
- The addressing scheme used in the penguin colony
- How "open" the Linux instances will be
- The connectivity requirements of your customers
- Isolation from other customers

In the following sections, we present some alternatives and describe "tricks" to using these alternatives (without going into too much detail). The objective is to enable technical readers who do not have a background in networking or routing to discuss issues with the networking people at their installation.

Routing domain

A *routing domain* is a group of networks that share routing information. Generally, all routers in a given domain can reach all other networks in that domain. Within a routing domain, interior gateway protocols are used to provide this sharing of routing information.

In Figure 4-7 on page 93, three networks are shown. These may be separate organizations, or divisions within a single organization. Each of the clouds represents a routing domain.

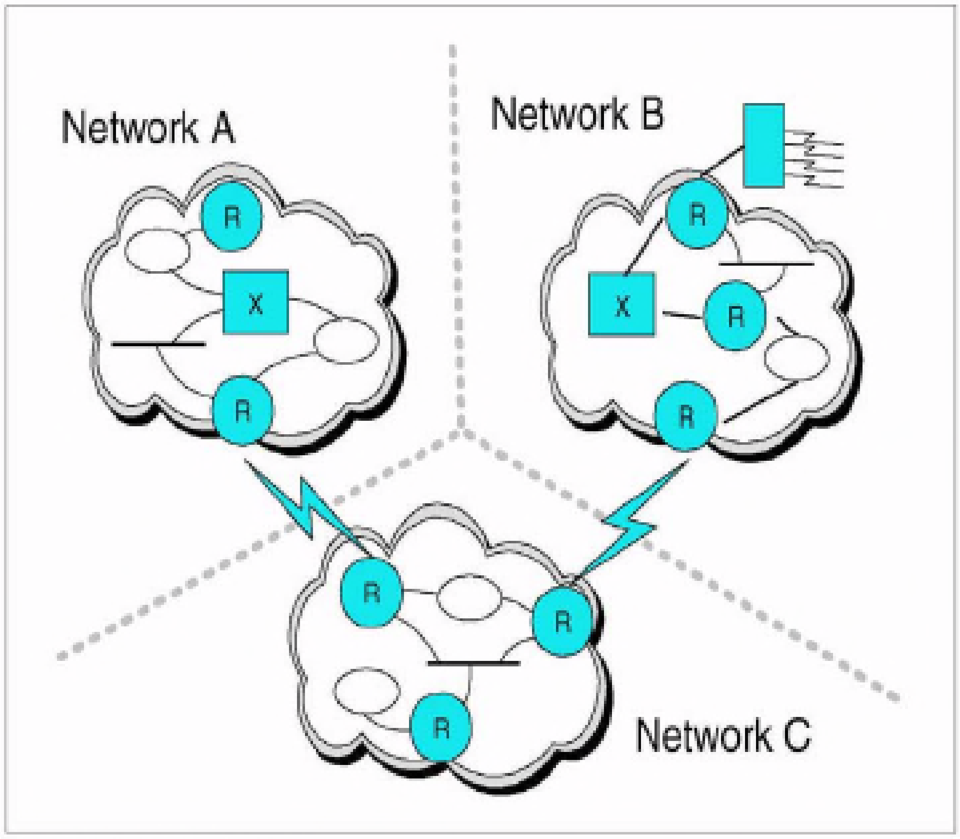


Figure 4-7 Routing domains

Routing domains can be used to control the flow of routing information in a large network, or between networks. For example, a large company with networks in cities all over the world does not need to have complete sharing of all routing information across all routers.

When devices in separate domains wish to communicate, some design work is required. Ideally, you want to allow connectivity as required, without exchanging topology information. An exterior gateway protocol is used in this role. Routers on either side inform each other about the networks that can be reached, without exchanging the detailed information provided in interior routing protocols.

In the example shown in Figure 4-7, control of the routing domains can allow communication between Network A and Network C, and between Network B and Network C, but prevent Network C from being used as a gateway between Networks A and B.

Choosing a routing protocol

This process cannot be done in isolation from the networking people (refer to "Talk to your friendly network staff" on page 70). As we have identified, for two-way communication to take place in a penguin colony, some interaction must take place between the penguin colony and the outside. If this boundary is not well planned, connectivity will be unreliable or may not even work at all. In the worst case, a poor design could cause problems in other, seemingly unrelated, parts of the network.

Example: Assume that a certain network chooses to use OSPF within its penguin colony. Because its router network also uses OSPF, it uses a different OSPF area number for the penguin colony. It sets up dynamic routing between virtual routers. The virtual routers that own the physical network connectivity for the penguin colony advertise themselves as the default routers for the area.

These outward-facing Linux systems would be the boundary between the penguin colony and the rest of the network. If they are not configured correctly, it could be possible for the Linux routers to become the default routers for the entire network! This could have catastrophic results on routing throughout the network.

One way of avoiding such problems is to use the *same routing protocol* inside the penguin colony as is used outside it. This has the benefit of uniformity across the whole network—and it does not necessarily mean that the penguin colony has to exchange complete network data with the rest of the environment (for example, OSPF allows different Autonomous Systems (AS) within a network, and this can be used to manage routing updates). This approach can be used in smaller networks, or where the consistency of using a single routing protocol is desirable.

Another method is to use a *different protocol* inside the penguin colony from the one being used outside, and have the first routers in the network manage the boundary processing.

Note: Most routers will maintain separate tables for each routing protocol. If a router running more than one routing protocol is properly configured, it can copy routing data from one routing domain into the other; this is known as "importing" routes.

For example, a router that has RIP configured on one interface and OSPF on another, can be configured to import the OSPF routes into RIP. This means that the routers in the RIP network will learn about networks in the OSPF domain through this router.

The third method is to use an *exterior gateway protocol* between the penguin colony and the rest of the network. This is an extension of the previous model, but has the effect of completely dividing the networks. It is useful when the penguin colony requires additional security, or if penguins from multiple customers are sharing infrastructure.

4.4.3 Using the OSA with Linux

Many different configurations are possible with OSA adapters, but basically one of two drivers will be used: For all OSA2 cards, the lcs.o driver is used; for OSA-Express fast Ethernet cards you have a choice: you can use the lcs.o driver in non-QDIO (or OSE) mode, or the combination of the qeth.o and qdio.o drivers in QDIO (or OSD) mode. For OSA-Express Gigabit Ethernet cards, the combination of the qeth and qdio drivers is used, because this card can only be run in QDIO mode.

Attention: In QDIO mode the OSA-Express card can communicate directly with the Central Processing Complex (CPC) using data queues in z900 memory and utilizing Direct Memory Access (DMA). This proves to be much faster than previous technologies. As this redbook was being completed it was noted that a small number of customers were concerned about performance of their OSA-Express Gigabit Ethernet cards while being driven by Linux z/VM guests. Through tuning, it was possible to significantly improve the performance of these cards. The tuning specifics will be incorporated as changes to the driver code in the near future. Before this driver code is available, the lcs.o driver with the OSA-Express fast Ethernet card in non-QDIO mode may give better performance on Linux guests under z/VM.

When installing distributions, the driver is chosen based on the type of networking you specify. We describe the files that are important for SuSE distribution in case you want to modify the networking driver implemented at install time:

- /etc/modules.conf - Here you associate the network drivers (e.g., lcs or qeth) with network interfaces (e.g., eth0 or tr0) via the `alias` statement. When using the 2.2 kernel, you also add the parameters for the appropriate network driver.
- /etc/chandev.conf - Here you specify the parameters for the appropriate network driver when using the 2.4 kernel.
- /etc/rc.config - For the SuSE distribution only, this is where many important networking variables are set. Specifically, these are as follows:
  - NETCONFIG - The number of network cards: (e.g. `NETCONFIG="_0"` for one, `NETCONFIG="_0 _1"` for two)
  - IPADDR_x - The TCP/IP address for each interface corresponding to _x (e.g., `IPADDR_0="9.12.6.99"` is the IP address for the first interface.)
  - NETDEV_x - The device or interface name for each interface corresponding to _x (e.g., `NETDEV_1="eth0"` is the device for the second interface.)
  - IFCONFIG_x - The parameters to be passed to the **`ifconfig`** command (e.g., `IFCONFIG_0="9.12.6.99 broadcast 9.12.7.255 netmask 255.255.255.0 mtu 1492 up"` are the **`ifconfig`** parameters for the first interface.)

For details on the drivers' parameters, see *Device Drivers and Installation Commands*. For 2.2 kernels, the manual can be found on the Web at:

`www10.software.ibm.com/developerworks/opensource/linux390/documentation-2.2.shtml`

For 2.4 kernels, it can be found on the Web at:

`www10.software.ibm.com/developerworks/opensource/linux390/documentation-2.4.shtml`

OSA Address Table (OAT)

The OSA has a unique design that allows it to be accessed by multiple LPARs or VM guests simultaneously, each with a different IP address, but attaching to the same physical LAN. This operating mode of the OSA is known as *shared mode*.

In shared mode, the OSA Address Table (OAT) is required to provide a mapping between host device addresses and network IP addresses. This allows the OSA to send IP packets to the correct LPAR or guest.

Note: For more information about OSA-Express, refer to the product manuals or the IBM Redbook *OSA-Express Implementation Guide*, SG24-5948, on the Web at:

http://www.ibm.com/redbooks/abstracts/sg245948.html

If your OSA is accessed *only* by the first-level router on behalf of your penguin colony, the OAT is not used. In this case, everything will work as you expect, since the OSA will forward all traffic to the VM guest that activates it, in exactly the way that a normal Ethernet adapter operates. If you are in shared mode, however, the OAT is used to determine which VM guest (or LPAR) will receive the incoming packet.

To ease the definition work required—and to simplify the configuration, in many cases—a particular VM guest or LPAR can be defined to the OSA as the *primary router*. In this case, all IP packets arriving from the network that do not have a corresponding specific OAT entry will be sent to the LPAR or guest nominated as the primary router. A secondary router can be defined, to which packets will be sent if the primary system is not active.

Note: Primary router is often referred to as *primary default entry* in OSA documentation. *Primary router* is the term that appears in Linux device driver documentation.

Figure 4-8 shows the process used to dispatch a packet through the OSA.

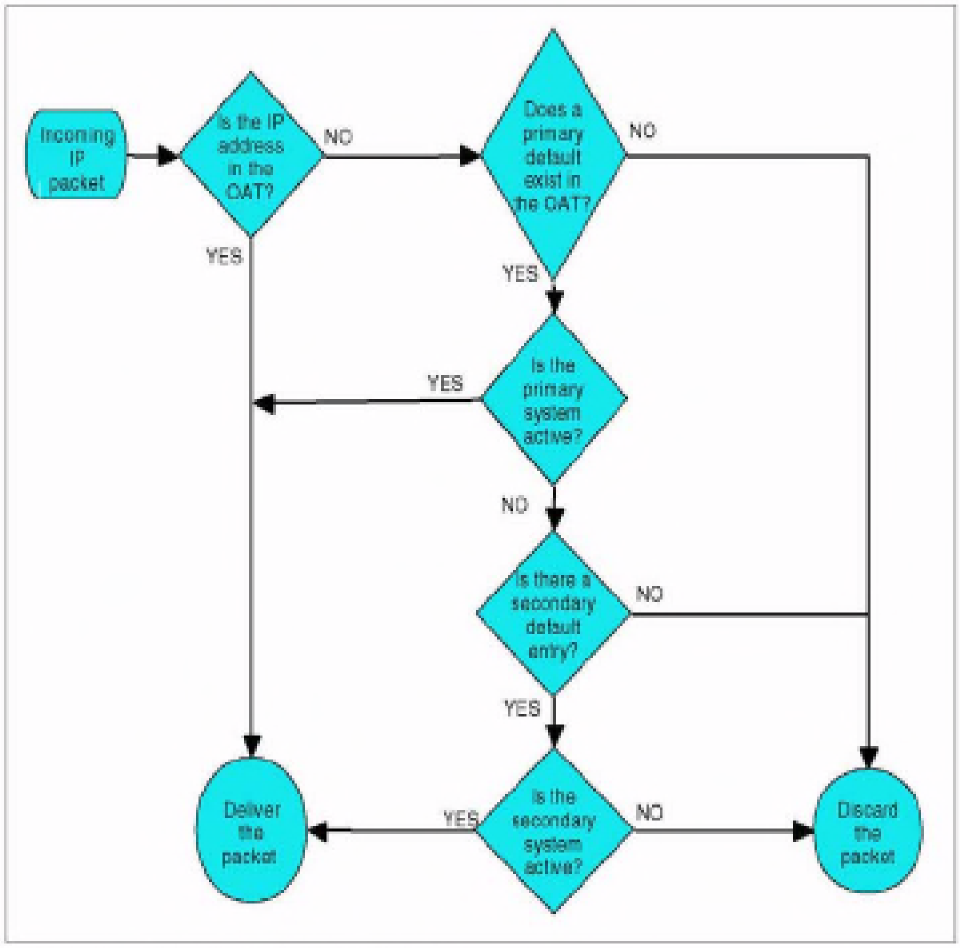

Incoming IP packet
Is the IP address in the OAT?
NO
YES
Does a primary default exist in the OAT?
NO
YES
Is the primary system active?
YES
NO
Is there a secondary default entry?
NO
YES
Is the secondary system active?
YES
NO
Deliver the packet
Discard the packet

Figure 4-8 Packet path through the OSA

The IP address used in this process is the destination IP address of the packet. This is may not be the IP address of your router system. If the packet is destined for one of the Linux images in your penguin colony, then it will be the IP address of that system and not the router's address.

Note: A complete description of how TCP/IP routing works is outside the scope of this book. Almost any documentation on Linux TCP/IP will provide an introduction to routing and TCP/IP connectivity. Alternatively, you can go to:

`http://kernelnewbies.org/documents/ipnetworking/linuxipnetworking.html`

The information at this URL provides a good insight into the way that TCP/IP is implemented in Linux.

To illustrate, in the following list we break down, step by step, the process involved in sending a packet from somewhere in the network to a Linux instance in a penguin colony, focusing on the steps that involve the OSA:

1. A packet leaves a client machine somewhere in the IP network. According to normal IP routing rules, the packet reaches the router immediately before the OSA in the path to the penguin.
2. The router, upon receiving a packet not intended for itself, has to route the packet onward through the network. It consults its routing table to determine the next router in the chain. It finds the correct route, which identifies your OSA-owning penguin as the next hop to the destination.
3. The routing table shows the IP address of the OSA-owning penguin. The router performs an Address Resolution Protocol (ARP) request to find the hardware address which matches the IP address in the routing table.
4. The OSA responds to the ARP request with its hardware address.
5. The router builds a new IP packet, with the same destination IP address field it has always had on its path through the rest of the network, and sends it over the Ethernet to the OSA.
6. The OSA receives the packet and consults the OAT to find out where to send it.

Note: This is the point at which the process can fail in the penguin colony environment. If the IP address of the Linux instance has not been added to the OAT, or if the OAT does not have a primary router defined, the OSA will throw the packet away, and the Linux router will not even be aware of that.

Important: Many network folk are surprised by this, since it is unusual to have an adapter make packet-forwarding decisions without input from the operating system. Due to the special capabilities of the OSA, however, this behavior is required.

The concept of the primary and secondary router in the OAT can be confusing. Primary router, in the OAT, does *not* mean that the OSA is routing packets. Think of the primary router as being the LPAR or guest that the OSA thinks is a router, and can handle packets unknown to the OSA.

Because each Linux instance in your penguin colony will have at least one IP address, the definition requirements for the OAT can be huge, or even impossible. OSA-2 could only have 16 entries in the OAT, with a maximum of 8 per LPAR or guest. This meant that only 7 Linux instances could be supported behind one router.

Even with OSA-Express, which raises these limits considerably, the definition workload would be enormous. Therefore, the current recommendation when using OSA adapters for penguin colony connectivity is to either define the router guest as the primary default entry in the OAT, or dedicate the OSA port to the router guest, using the OSA non-shared.

Using multiple OSAs

In order to support a high-availability configuration, you may choose to provide more than one OSA to your Linux router. This usually involves connecting each OSA to a different LAN, with redundant network connectivity.

You may also use multiple OSAs to provide greater bandwidth to your penguin colony, in which case you can have these OSAs connected to the same LAN. In this configuration, you need to be aware that the way the Linux kernel responds to ARP requests may cause connectivity to fail if you are not using primary routers in your OSAs.

When an ARP request is received by Linux, the kernel checks to see if the request is for *any* interface on that machine. If it is, then the adapter that received the request will respond to the ARP request with its hardware address, even if the IP address belongs to another interface.

Note: This behavior is generally an efficiency and availability advantage. The first adapter that receives the ARP request is possibly less busy than other adapters, and it makes sense to respond to the request as soon as it is received, rather than waiting until the request is received at the correct adapter. There may also be a physical cabling problem with the other adapter, in which case it would never respond.

This behavior will not work with the OSA, however, since only the OSA with the IP address defined will be able to pass traffic for that address. If you are not defining your Linux router as the primary default entry in the OAT, you must turn off this behavior in Linux with the following commands:

```
echo 1 > /proc/sys/net/ipv4/conf/all/hidden
echo 1 > /proc/sys/net/ipv4/conf/<device-name>/hidden
```

Note that the line specifying the device name must be repeated for *each* OSA interface. These commands must be executed when the system is started, so adding them to the network startup script is most efficient.

QDIO mode

OSA-Express adapters can be defined to support QDIO. This mode gives high-speed operation through direct use of the STI bus, but it also provides a facility which allows the operating system to dynamically update the adapter's OAT. This dynamic update happens using an "out-of-band" signalling path between the driver and the adapter.

Note: "Out-of-band" signalling is a telecommunications term which refers to a signalling or control channel that is separate from the path used for data flow. In the case of the OSA-Express, this path is an extra device address required when using QDIO mode instead of LCS mode.

In QDIO mode, the OAT does still exist, even though you no longer have to build it manually. Under Linux, the qeth module sends the required information to the OSA-Express to allow it to update the OAT. Setting primary router status is also controlled dynamically through the qeth driver.

OAT update from VM TCP/IP or CS/390

QDIO support in VM TCP/IP and Communications Server for OS/390 provides updates to the OAT for any configured interface. This means that not only is the IP address for the OSA adapter itself added, but any other interfaces, as well. This support is required particularly for OS/390, where the use of Virtual IP Addressing (VIPA) required the VIPA address for a stack be added to the OAT.

There are no updates for addresses that are not connected to the stack, however. This means that, in the case of an IUCV connection under VM TCP/IP, for example, the IP address of the local end of the connection is added, but the remote end is not.

OAT update from Linux

Under Linux, the qeth driver will only add the OSA's own address to the OAT. No other addresses on the Linux system are added.

However, the qeth driver supports an interface through the /proc file system which can be used to manually add an OAT entry against this instance of the qeth driver. This would allow logic to be added to the network configuration scripts for CTC and IUCV interfaces. When the interfaces are configured, the script would additionally invoke the qeth driver to add the peer IP address for the link to the OAT.

Sharing an OSA

For some time, it was considered unsafe to share an OSA adapter between OS/390 or VM systems and Linux. Now, with recent versions of the LCS driver, this precaution is not necessary. However, you may still choose to exercise caution when sharing with high-profile production systems.

The choice of whether to share or not to share will become part of your design process. For example, an existing z/OS installation using one port on an OSA-Express Fast Ethernet card may have set primary router on a z/OS LPAR to support the Application VIPA function in z/OS. In this configuration, the installation is unable to have a z/VM LPAR configured as primary router to support a penguin colony under z/VM, since an OSA port cannot have more than one primary router specified. In this situation, two possible solutions exist:

- Use the second OSA-Express Fast Ethernet port.

  One port could be dedicated to Z/OS as per the current scenario, and the second port could then be dedicated to z/VM to be used for the penguin colony. Each port can have primary router specified independently.

- Use Queued Direct I/O (QDIO) mode in the OSA-Express.

  In this mode, z/OS can update the internal OAT in the adapter, as required. This eliminates the need for primary router to be set on the z/OS LPAR, so that z/VM can be given the role on behalf of the penguin colony. A CTC connection could be set up between z/VM and z/OS to route any traffic intended for z/OS that goes to z/VM incorrectly.

There is also a third solution which is to route all traffic through the z/OS system, and use a CTC connection from there to the z/VM system. This solution does increase the hop count to the Linux systems. Also, many z/OS installations prefer to keep z/OS TCP/IP traffic separate, for security reasons. On the positive side, z/OS has a new feature called *HiperSockets Accelerator* which could prove useful in such a configuration.

4.5 Other issues

In this section, we address CTC/IUCV device restrictions and a VM TCP/IP restriction.

4.5.1 CTC/IUCV device restrictions

The following device restrictions exist.

Numbers of links

When building a large penguin colony and using a virtual routing scenario, it is desirable to have as many Linux instances as possible supported off one router. This increases the server-to-infrastructure ratio of your penguin colony. Unfortunately, there are currently some restrictions in place that limit the number of links that can be created.

Attention: The following points describe ways that the kernel support for CTC and IUCV can be modified to increase the number of devices supported. In order to make these changes, you *must* be familiar with kernel recompilation and C language syntax. Also, using a rebuilt kernel may have ramifications if you have a support agreement.

In the Linux 2.2 kernel, the CTC driver has a hard restriction of a maximum of 8 links. A `#define` directive in ctcmain.c sets this, and it could be increased to allow more CTC links to be set up.

In the 2.4 kernel, the restriction has changed: when CTC is compiled into the kernel, a restriction of 16 links applies, but when the driver is modularized there does not appear to be a restriction.

The IUCV driver appears more friendly to increasing the number of devices. While both the 2.2 and 2.4 kernel levels have `#define` directives limiting the maximum number of devices to 10, the later versions of the driver provide instructions in the code for how to increase the limit.

Attention: Modifications to these drivers should not be considered lightly; issues such as memory consumption must be taken into account. For example, the CTC driver allocates a 64 KB buffer for each connection it manages. With 256 connections, a total of 16 MB of memory would be locked in the CTC driver alone! Therefore, we reiterate: modify these drivers at your own risk.

Configurability

As mentioned in "Linux as a virtual router" on page 68, the CTC and IUCV drivers cannot currently be configured dynamically. This means that the modules must be unloaded and reloaded to support newly added devices, which requires all interfaces using that module to be shut down.

This is because the drivers currently only scan for valid devices when they are initialized. The drivers build control blocks and buffers at that time, and are not designed to support the dynamic addition of new devices.

A useful enhancement to the CTC and IUCV drivers would be the ability to perform this dynamic reconfiguration. The kernel already provides support to recognize newly configured devices, and the DASD driver is also being enhanced in this area. Reconfigurable CTC and IUCV drivers would allow a much more flexible penguin colony environment to be created, where less planning of future connectivity would be required—and adding a new penguin to the colony would be much easier and more dynamic.

4.5.2 VM TCP/IP restriction

Currently, performance of a VM TCP/IP stack degrades after approximately 100 DEVICEs have been defined. The details on this are not clear, and we are not aware of the nature of the degradation in performance. We are also not aware of any active work to research this further.

We mention this as a prompt for you to find out more, so that you can factor this into the planning for your penguin colony.

5

Security architecture

In this chapter we discuss security topics related to running Linux on a S/390 or zSeries. Running a Linux system requires reliable security on any hardware platform. Using hardware other than S/390 or zSeries often leads to discrete servers with dedicated hardware resources.

With Linux on S/390 or zSeries, together with the partitioning of the hardware and the virtualization offered by VM available to the Linux systems, many new aspects of security arise. On one hand, you must consider if and how the physical resources of the S/390 or zSeries hardware will be isolated from the virtual Linux systems. On the other hand, the maintenance of multiple Linux systems with VM requires some additional security features in front of the penguins, but also offers clever methods of handling a colony of them.

5.1 Sharing isolation and reconfiguration of resources

As discussed in Chapter 1, "Introduction" on page 3, Linux can run on zSeries and S/390 hardware in three basic modes:

- In native mode
- In LPAR mode
- As VM guests

In the following sections, we discuss security as it pertains to running Linux in the three different modes.

5.1.1 Running Linux in native mode

The simplest security scenario is with Linux running in native mode. Here Linux is the only operating system on the hardware. Therefore, all internal resources (processors, memory, channels) are dedicated and there are no concerns about sharing these resources with other systems. It is still possible to share external resources such as DASD with other mainframe systems, but that would imply the same problems as if the Linux systems were running on a hardware platform other than S/390 or zSeries.

5.1.2 Running Linux in LPAR mode

The security scenario with Linux running in one or more LPARs gets more complicated because the PR/SM microcode is used to share the S/390 or zSeries resources.

An independent operating system can run in each LPAR (whether it's Linux, OS/390, VM or others). These LPARs and the various operating systems can share internal resources. From a logical point of view, the resources seem to be dedicated to the partition—but from a physical point of view, the hardware is virtualized and partitioned by PR/SM to be shared among up to 15 LPARs. PR/SM ensures that each LPAR uses only the resources that have been allocated to it. If total isolation of a partition is required, the system can be configured via the IOCP and IOCDS so that no resources are shared among another partition.

An ISP or ASP has to deal with various servers from several customers, who all have an essential interest in separating their data and applications from those of other customers. To make the principle of virtualization by PR/SM acceptable for these customers, some questions have to be answered:

1. Is there an official certification that the processes of the PR/SM microcode are secure, especially under the new zSeries architecture, which adds new features such as HiperSockets?
2. If CPs are shared between LPARs, is there any risk that one LPAR can influence the operations in another LPAR?
3. What is the risk if memory is reconfigured from one LPAR to another?
4. How can the dedicated and secured use of peripheral devices be assured, if shared channels and control units are used?
5. Should there be a secure administration access to Linux for zSeries and S/390 systems, especially to those that are logically placed in DMZs?

PR/SM certification

Question 1 asks for official statements and guarantees of the security provided by hardware and microcode. For S/390 servers, which means up to the 9672 G6, PR/SM has been evaluated according to the European ITSEC E4 (Information Technology Security Evaluation Criteria, evaluation level E4). This evaluation certifies that PR/SM can separate and isolate partitions as if they were running on physically separate systems. This includes topics such as the following:

- Identification and authentication
  - PR/SM will associate a unique identifier with each logical partition in the current configuration.
  - Each LPAR is uniquely identified, based on IOCDS definitions.
  - The identifier is used to mediate access control.
- Audit and accountability
  - All security relevant events are recorded in an audit log.
  - The audit log is protected from unauthorized deletions or modifications.
  - Applications in LPARs cannot read the audit log.
- Access control
  - LPAR security controls define a partition's access to IOCDSs, performance data, crypto, and reconfigurable channels.
  - Access to control units and devices on shared channels can be restricted.
  - Dedicated channels, storage, and CPs are never shared.
  - PR/SM will prevent the transfer of a message between a logical partition and any resource not explicitly allocated to it.
- Object reuse
  - Storage will be cleared prior to allocation or re-allocation.
  - All information in physical processors or coprocessors will be reset before dispatching the processor to a new logical partition.

  – Non-shared channel paths and attached I/O devices will be reset prior to allocation to a LPAR

For zSeries servers, a certification based on Common Criteria EAL5 is being worked on. This certification uses roughly the same criteria as the older ITSEC E4. For more details, see the URL:

`http://www.commoncriteria.org`

To answer questions 2 through 5, we have to take a closer look at the way PR/SM handles the sharing, isolation and reconfiguration of resources among LPARs.

Processors in a LPAR environment

LPARs running on S/390 or zSeries hardware can use either dedicated PUs (which only one LPAR is allowed to use), or shared PUs (which every LPAR defined for shared CPs can access).

CPs or IFLs (but not both, in the same LPAR) can be used to run Linux workloads. A CP is a general purpose processor that has the full ESA/390 and z/Architecture instruction sets, and it can run all operating systems available for S/390 or zSeries, including OS/390, z/OS, VM/ESA, z/VM, VSE, and Linux. The IFL can only be used to run native Linux, or z/VM V4 with Linux guest systems. While the number of CPs determines the model of the 9672 or 2064 (and with this the base measurement for the software licenses running on this server), an IFL is not counted in the pool of PUs that determine software pricing.

While an OS/390 or z/OS LPAR requires CPs, a Linux LPAR (which means native Linux, or z/VM V4 with Linux guest systems; VM/ESA LPARs still require CPs) can run either on CPs or on IFLs. Therefore, usually OS/390 or z/OS LPARs (which require CPs) do not share processors with native Linux or z/VM LPARs (which can use IFLs). But there still is the possibility, especially for small testing LPARs, to run Linux on a CP shared with a OS/390 or z/OS LPAR. And of course IFLs can be shared between two or more Linux LPARS, but not between Linux and z/OS LPARs.

When a processor is shared between two or more LPARs, each partition is given access to this processor for a certain amount of time. This is controlled by an algorithm based on the defined weight of the LPARs. Different partitions may use a single physical processor at different times, but the information held in the processor will be cleared before it is used by another partition.

If the time allocation for the LPAR is used up, or when the LPAR has no more work to do for the processor, or when a higher prioritized LPAR is demanding the processor, an interrupt occurs and the access to the processor is given to the other partition. When a processor is switched from use by one LPAR to another, its entire state is preserved and the new LPAR's state is restored. For the duration of its timeslice, only this LPAR has access to the processor.

Memory in a LPAR environment

Memory is not shared between LPARs, but the total amount of memory installed on a 9672 or 2064 is divided among the LPARs. Logical storage, both central and expanded, is represented by the same amount of contiguous physical storage of the same kind. This is done when the LPARs are activated, and PR/SM does not move logical partitions once they have been placed in real storage.

PR/SM also allows that physical storage can be deallocated from one logical partition and reallocated to another while these LPARs are running. If reconfigured, the storage is cleared as part of the reconfiguration process, so no data can be presented from one LPAR to another.

However, the operating system has to support this reconfiguration by being able to clear the areas of memory that are scheduled to be configured off this LPAR and given to another. VM or native Linux do not support dynamic storage reconfiguration, so there is no security impact of running Linux on S/390 at all.

Channels and external devices in a LPAR environment

The S/390 and zSeries architectures allow channels and the external devices connected to these channels to be shared between multiple LPARs.

It is also guaranteed that channels dedicated to one LPAR cannot be accessed by another LPAR, that access to control units and devices on shared channels can be restricted, and that non-shared channels will be reset before reallocation to other LPARs.

If a device that has to be accessed using a shared control unit and shared channels should be dedicated to one LPAR, this has to be defined in the I/O Configuration DataSet (IOCDS). PR/SM ensures that no LPAR can access a device that is not defined to this partition. It prevents the transfer of data between a LPAR and any resource not explicitly allocated to it.

In an environment with several Linux systems running in different LPARs, it is possible, and in most cases, sensible, to share channels and control units between the LPARs—and thus between the Linux systems. However, you need to be careful when DASD devices are shared. Unlike OS/390 or z/OS, the Linux operating system itself does not have any built-in procedures that prevent system images from interacting with each other's writing operations and destroying the integrity of the data. Unless the applications using the DASD devices do provide special procedures to ensure data integrity, write operations to a shared DASD device should only be permitted on *one* Linux system, with the others only allowed to read the data.

Networking in a LPAR environment

Network devices are handled the same way as other external devices. They are either channel-attached control units, or located on Open Systems Adapter (OSA) cards. The ports on the OSA cards can be shared between LPARs, just as any other channel. The security in shared usage, isolation and reconfiguration of these devices is the same as with other channels and control units. Additionally, the network outside of the physical S/390 or zSeries server has to be designed and set up in a way that no unauthorized access to data or resources is possible.

For system administration tasks especially, secure network access has to be granted, and this administration network should be separated from the normal Intranet and Internet.

For the internal network communication between LPARs on the same zSeries server, two choices are provided which do not need to use any hardware outside. A TCP/IP connection between two LPARs can be set up by using a shared OSA port, or by using an in-memory communication vehicle called HiperSockets.

The first environment to support HiperSockets will be Linux for zSeries, z/VM, z/OS, and any combination of these operating systems running in LPARs on a zSeries server (there are no HiperSockets for 9672 G5/G6 and Multiprise 3000 servers). Because these connections are not physically attached to the outside network, they are as secure as the LPARs themselves.

5.1.3 Running Linux under VM

Similar security questions arise with the usage of z/VM or VM/ESA to virtualize the resources of an LPAR:

- How are VM resources and definitions protected against guest systems?
- What is the remaining risk if the resources of VM guest systems (memory, CPs) are reconfigured?
- How secure are the different kinds of communication among Linux images (for example, OSA, HiperSockets, Guest LAN, IUCV or VCTC)?
- How can the dedicated and secured use of peripheral devices be assured, if shared channels and control units are used?
- How can it be proven that VM guest systems are isolated from each other?

The System Integrity Statement for VM

The last question is asking again for official statements and guarantees for the security provided by VM. There is no official certification of VM/ESA or z/VM comparable to the ITSEC E4 certification of PR/SM. However, the z/VM guest machine separation uses the very same machine facilities that were created for, and are used by, PR/SM. So the same level of trust can be placed in z/VM and VM/ESA guest machine separation as in the PR/SM microcode.

Furthermore, IBM gives a System Integrity Statement for z/VM (in the publication *z/VM General Information*, GC24-5991), which is cited here:

> System integrity is an important characteristic of z/VM. This statement extends IBM's previous statements on system integrity to the z/VM environment.
>
> IBM has implemented specific design and coding guidelines for maintaining system integrity in the development of z/VM. Procedures have also been established to make the application of these design and coding guidelines a formal part of the design and development process.
>
> However, because it is not possible to certify that any system has perfect integrity, IBM will accept APARs that describe exposures to the system integrity of z/VM or that describe problems encountered when a program running in a virtual machine not authorized by a mechanism under the customer's control introduces an exposure to the system integrity of z/VM, as defined in the following "z/VM System Integrity Definition" section.
>
> IBM will continue its efforts to enhance the integrity of z/VM and to respond promptly when exposures are identified.

After this statement, in which IBM guarantees fixing every exposure of the system integrity of z/VM, there follows the "z/VM System Integrity Definition":

> The z/VM control program system integrity is the inability of any program running in a virtual machine not authorized by a z/VM control program mechanism under the customer's control or a guest operating system mechanism under the customer's control to:
>
> – Circumvent or disable the control program real or auxiliary storage protection.
> – Access a resource protected by RACF. Resources protected by RACF include virtual machines, minidisks, and terminals.
> – Access a control program password-protected resource.
> – Obtain control in real supervisor state or with privilege class authority or directory capabilities greater than those it was assigned.
> – Circumvent the system integrity of any guest operating system that itself has system integrity as the result of an operation by any z/VM control program facility.

> Real storage protection refers to the isolation of one virtual machine from another. CP accomplishes this by hardware dynamic address translation, start interpretive-execution guest storage extent limitation, and the Set Address Limit facility.
>
> Auxiliary storage protection refers to the disk extent isolation implemented for minidisks/virtual disks through channel program translation.
>
> Password-protected resource refers to a resource protected by CP logon passwords and minidisk passwords.
>
> Guest operating system refers to a control program that operates under the z/VM control program.
>
> Directory capabilities refer to those directory options that control functions intended to be restricted by specific assignment, such as those that permit system integrity controls to be bypassed or those not intended to be generally granted to users."

This definition, together with the preceding statement, is a guarantee that VM is able to provide full system integrity to the VM guest systems, and that IBM will fix any exposure to this. However, because the CP program and the guest systems are under the control of the customer, the achieved level of system integrity depends on the way the VM environment is set up and maintained. This also is made very clear by having customer responsibilities being defined as follows:

> While protection of the customer's data remains the customer's responsibility, data security continues to be an area of vital importance to IBM. IBM's commitment to the system integrity of the z/VM environment, as described in this statement, represents a further significant step to help customers protect their data.
>
> Product documentation, subject to change, describes the actions that must be taken and the facilities that must be restricted to complement the system integrity support provided by z/VM. Such actions and restrictions may vary depending on the system, configuration, or environment. The customer is responsible for the selection, application, adequacy, and implementation of these actions and restrictions, and for appropriate application controls.

So, to give a short answer to the question: there is no external proof or certification (like E4 from the ITSEC for PR/SM) that VM systems are isolated from each other. But IBM warrants the integrity of the virtual machine interface and will accept integrity APARs and fix any problem exposed.

To answer the other questions, again we take a closer look at the way VM handles the sharing, isolation, reconfiguration, and management of resources between guest systems. A comprehensive summarization for this is given in Appendix B of *Linux for S/390*, SG24-4987.

Definition and management of guest systems

Simply put, z/VM transforms the principles of partitioning—which on the hardware level are provided by the PR/SM microcode—to the LPAR environment, and enriches them with virtualization. The Control Program (CP) of VM is able to virtualize hardware resources, either by sharing or partitioning real hardware resources, or by emulating their behavior. The definition of the virtual guest systems and of the resources available to them, as well as the management of this environment, is also provided by the CP.

Operating system failures that occur in virtual machines do not usually affect the z/VM operating system running on the real processor, and or the other guests. If the error is isolated to a virtual machine, only that virtual machine fails and the user can re-IPL without affecting the testing and production work running in other virtual machines.

VM resources and definitions are protected through *privilege levels*. A guest can, in general, manipulate its own environment; but without special privileges, which are under customer control, one guest cannot manipulate another's environment. Users of the virtual machines are unaware of the virtualization done by CP, just as an LPAR is unaware of the virtualization done by PR/SM.

User access to the VM system and its virtual machines can be controlled by the Resource Access Control Facility (RACF) licensed program, the strategic security facility for VM/ESA and z/VM. RACF also checks the authorization for the use of system resources like minidisks and data in spool files, and audits the use of them.

However, the RACF database cannot be shared with OS/390. In addition, in a complex VM environment, we recommend that you use the Directory Maintenance (DirMaint) product to maintain the user directory.

Processors in a VM environment

The VM Control Program defines and assigns logical processors to the guest systems, the virtual machines. These logical processors are matched to the logical processors of an LPAR (or to the physical processors, with VM running on the native hardware), which PR/SM maps to shared or dedicated physical processors.

A virtual machine can have up to 64 virtual processors defined, although a zSeries server can physically only have 16 processors (CPs or IFLs). If the operating system running in the virtual machine is capable of using multiple processors, it will dispatch its workload on its virtual processors as if it were running in a dedicated hardware environment.

The VM Control Program handles dispatching the virtual processors on the real processors available to that virtual machine. A real processor can either be dedicated to a virtual machine, or shared among virtual machines. (Keep in mind that the VM CP only handles the processors it controls, which means that if VM is running in an LPAR, a real processor to the VM Control Program can also be a shared physical processor dispatched by PR/SM.)

There is no security risk if the resources of VM guest systems are reconfigured, or if the virtual processors of different guest systems are dispatched to the same physical CPs or IFLs. The state of a processor is preserved for one guest and restored for another by the VM Control Program (just as PR/SM does for LPARs). Therefore, no information can be accidentally passed from one VM guest system to another in this way.

Memory in a VM environment

Each virtual machine has its own defined virtual memory. The physical residency of the guest system's memory pages in real storage is managed by the VM Control Program's paging mechanism. Pages that have not been referenced can be moved out of real storage, either into expanded storage or onto a paging device.

When a virtual machine touches a page that is no longer in real storage, a page fault occurs and the Control Program will bring the missing virtual page back into real storage. The memory addresses used within a virtual machine are also virtual, and they have no meaning outside the virtual machine.

The VM Control Program also allows the sharing of virtual pages by a number of virtual machines. A shared virtual page requires just one page of real storage, no matter how many virtual machines are sharing it. This can be used for sharing the Linux kernel, which is read-only to the guests, enforced by the hardware.

Virtual disks (VDISK) can also be shared by several virtual machines—and data from shared minidisk (MDISK) caches can be copied to private virtual pages. This can have a profound effect on the productive use of multiple cloned Linux guest systems in a z/VM environment. Refer to Chapter 3, "Virtual server architecture" on page 45 and Chapter 10, "Cloning Linux images" on page 209 for a more detailed discussion.

There also is no security risk if the memory of VM guest systems is reconfigured, or if portions of the virtual memory of one guest are located by the CP in physical memory regions where the data of another guest resided earlier. Memory is cleared when it changes hands, and there is no vestigial information that can "leak" from one guest to another.

Channels and external devices in a VM environment

Each operating system running in its own virtual machine communicates with virtual devices. The mapping of virtual to real devices and resources is handled transparently by the VM Control Program.

The virtual DASD devices used by virtual machines are called VM minidisks. They are implemented by partitioning a real S/390 volume into cylinder ranges that appear as separate disk volumes to the virtual machine. A minidisk can span a whole real disk volume. A real disk can also be dedicated to a virtual machine.

Minidisks (MDISK) can be shared or non-shared. If authorized, one virtual machine can link to a minidisk belonging to another virtual machine to access the data on it. Links can either be read-only or read-write.

When a minidisk is write-shared, some software is needed to manage access to the data. CP is able to cache the contents of minidisks in real or expanded storage to improve application response times, and to share this minidisk cache between several virtual machines.

It is also possible to define virtual minidisks (VDISK), which are mapped into real storage by the VM Control Program, instead of residing on real DASD volumes, and to share them. The principles of using shared minidisks, shared minidisk caches and shared virtual minidisks between multiple Linux guest systems is also very important for running multiple cloned Linux guest systems under z/VM; refer to Chapter 3, "Virtual server architecture" on page 45 and Chapter 10, "Cloning Linux images" on page 209 for a more detailed discussion.

If devices such as an OSA port are dedicated to a VM guest, the VM operating system does not influence the use of this device by the guest operating system. Also, the dedicated and secure use of peripheral devices such as VM minidisks is assured, even if shared channels and control units are used.

From a VM perspective, physically shared but logically distinct devices (e.g. minidisks) are, for all intents and purposes, separate. One guest cannot access another's data (e.g., by seeking beyond the end of their disk). Interference by one guest with another from a performance viewpoint can occur, but it is controlled by VM scheduling mechanisms. Where devices are logically shared (e.g., a shared minidisk), authorization must be given by a system administrator to establish the sharing.

Networking in a VM environment

Network communication between a VM guest system and the outside world is established over the same physical hardware devices (OSA, channel-attached control units) as previously described, but the VM Control Program manages access to them. Of course, VM only manages the devices when they are defined as shared (rather than dedicated to only one virtual machine). The network has to be designed and set up so that no unauthorized access to data or resources is possible—and for system administration tasks, a separate network with secure access is especially recommended.

For network communication between a virtual machine in a VM LPAR and another LPAR on the same zSeries server, a shared OSA port can also be used. If both operating systems support this connection, even HiperSockets can be used (at the time writing, however, no Linux driver for HiperSockets was available).

For communicating between two virtual machines running in the same VM system (with VM running in an LPAR or on the native hardware), three additional communication vehicles are available:

- The virtual Channel-To-Channel (VCTC) device uses virtual I/O instructions. These are intercepted by the VM Control Program, which moves the data between memory buffers.
- The Inter-User Communications Vehicle (IUCV) provides a high-speed pipe for communications between virtual machines and the VM TCP/IP stack. IUCV connections can be established between pairs of virtual machines on the same VM system, or even on different VM systems.
- While VCTC and IUCV offer point-to-point connections, the VM Guest LAN, introduced with z/VM 4.2, provides multipoint virtual connections between guests, using virtual HiperSockets adapters within a z/VM system.

The VCTC and the even faster IUCV and Guest LAN connections are essential for the network design of multiple Linux virtual machines running in one VM system, with VM running both in LPAR or in basic mode.

All the different kinds of communication between guest systems and LPARs (such as shared OSA, HiperSockets, IUCV or VCTC) are completely secure in that an unauthorized third party cannot eavesdrop on them. However, keep in mind that access to these connections is only as secure as the connected operating systems using them.

5.2 Linux security

It is beyond the scope of this redbook to describe all the methods and tools available to increase security on Linux. In this section we discuss general recommendations for keeping your Linux server secure. In "Additional security documentation" on page 120, we list other sources you can refer to get a more comprehensive overview of the various security issues surrounding a Linux installation.

5.2.1 What kind of protection do you need

The way a Linux server should be protected is highly dependent upon the server's purpose. Therefore, consider what kind of access to the server is required, what exposures have to be taken into account, what kind of security attacks can be expected, and which tools to use in order to ensure the security of the system.

Based on the security basics summaries offered in *Linux for S/390, SG24-4987*, and in *Linux for zSeries and S/390: Distributions*, *SG24-6264*, we make the following general recommendations for protecting your Linux server:

- Disable unneeded services

  Depending on the Linux distribution used, different services are activated by default. Many of the network-related services are handled by `inetd` (or `xinetd`). You can deactivate many of these services by editing the /etc/inetd.conf or etc/xinetd.conf files (but you should consider carefully if the services are really unneeded before you remove them).

- Use the tcp wrapper

  To protect and log the remaining services, the tcp wrappers daemon (`tcpd`) should be used. When a communications service request is received, `inetd` invokes `tcpd` first, which then can invoke the requested service. The `tcpd` daemon performs some verification, logs the request according to the settings in the /etc/syslog.conf file, enforces access control and then, if everything is in order, passes the request on to the requested service.

- Use Secure Shell for remote access

  Simply stopping a service like telnet is not a good solution for a Linux server that needs to be accessed remotely. To allow remote access and to prevent the password sent to telnet from being "sniffed", replace the telnet service with the Secure Shell (SSH).

  SSH connections are encrypted and protected against IP or DNS spoofing. Similarly, the secure copy (`scp`) command can be used instead of FTP, and secure login (`slogin`) can be used instead of rlogin.

  For additional security, remote login for root can be forbidden. Then root access will be limited to the Linux console which is a VM session. In this scenario, both the VM and the Linux passwords would have to be cracked.

- Use shadow password utilities

  The /etc/passwd file often contains encrypted passwords that can be read by all users. This creates the possibility that weak passwords can be cracked via dictionary attacks. To avoid this vulnerability, we recommend that you use the shadow password utility, where passwords are stored in the /etc/shadow file (which does not have read access).

Additionally, this file contains information about expiration and renewal, so maintenance of passwords is eased. Use of shadow passwords is the default in current versions of all major distributions.

- Use the Pluggable Authentication Module (PAM)

  The PAM provides a library of authentication modules, all located in the /etc/security directory. These modules offer standard procedures for authentication purposes and can be used by various services, which configuration files are listed in the directory /etc/pam.d.

  Without these modules, every application would have to contain its own authentication code, if anything more than the standard user authentication by password is required. And if the authentication requirements of an application change, the whole application would have to be recompiled. By using PAM, only the configuration file in the /etc/pam.d directory has to be changed.

- Monitor security news and alerts

  In order to keep abreast of news concerning vulnerabilities or bugs in software running on the Linux server, the system administrator should check the Web sites related to these products often, and also check general Linux security-related URLs.

  In addition, the sysadmin should frequently monitor the log files of the applications (usually located in the directory /var/log/) for any problems.

- Use LDAP servers for authentication

  For directory services over TCP/IP, the Lightweight Directory Access Protocol (LDAP) should be used. One or more LDAP servers contain the data making up the LDAP directory tree. Information about objects (such as people, organizational units, printers, documents, etc.) are stored as entries in a hierarchical structure. LDAP provides a mechanism for clients to authenticate to a directory server, in order to get access to the information provided there.

  The implementation usually provided with Linux is OpenLDAP; refer to the following URL for more information:

  `http://www.openldap.org`

  Setup and usage of OpenLDAP is described in Linux for zSeries and S/390: Distributions, SG24-6264. OpenLDAP provides a directory service called **slapd**, which handles access control to, and management of, the databases containing the directory trees. It also contains the **slurpd** daemon, which helps **slapd** provide replicated service.

- Use firewall tools to secure your network

To control the traffic of TCP and UDP packets by using the IP firewall rules on the Linux kernel, IPTables should be used; see Chapter 11, "Network infrastructure design" on page 235 for details. Thorough evaluation and planning of the network infrastructure is required regarding the setup of Virtual Private Networks (VPN), including the arrangement and use of proxies, reverse proxies, and firewalls.

- Protect against viruses and trojan horses

  Because of its software architecture (in particular, memory management and file/user permission design), Linux is not susceptible to the traditional viruses that plague more elementary operating systems like Windows.

  But this does not mean that Linux is entirely safe from mischief or external threats, especially "trojan horse" programs. The distinction between a virus and a trojan horse is critical, and illustrates why Linux is relatively immune to viruses but not to trojan horse programs—so let's explain these terms here.

  What is a virus

  A *virus* is a program that actively operates, independently of a user, to attack various system resources. On most Windows systems, a user is also the administrator. Therefore, all system resources (disk, memory, applications, files, logs, devices, etc) are accessible by anyone or anything, including the virus program.

  It is impossible for a Linux operating system to suffer system-level damage from a virus, because it cannot get access to low-level system functions. However, just because Linux is relatively safe from viruses doesn't preclude it from spreading mail-based viruses when it is being used as a mail server.

  For this reason, there are antivirus programs that can be used for detection and disinfection of viruses and malicious code passing through Linux firewalls; for example, refer to the following URL:

  http://www.f-secure.com

  What is a trojan horse

  By contrast, a *trojan horse* is a program that cannot operate unless it is invoked (unwittingly) by a user. Generally speaking, Linux systems don't execute trojan horses on their own; they must be executed explicitly by the user, and are especially dangerous if the user is the root or superuser of a Linux system.

  Therefore, to help prevent the introduction and execution of a trojan horse, a system administrator should, at a minimum, avoid logging in as root or otherwise assuming superuser capability unless it is absolutely necessary for some sysadmin task, and should furthermore ensure proper access permissions on files (which is particularly important for system utilities, devices and log files).

A very useful hardening tool is tripwire, which is able to detect and report file and directory modifications. This can help to detect trojan horses and modified software left by hackers, for example for sniffing passwords.

- Use tools for testing network security

  You can use tools such as the `scanlog` daemon to test network security. This daemon is able to recognize if someone is requesting more than a specific number of ports per second, which can indicate that someone is scanning the Linux server for insecure ports.

5.2.2 Additional security documentation

The following documentation provides detailed information about Linux-related security topics.The paper "Addressing Security Issues in Linux", by Mark Chapman, presents a broad overview of the various security issues regarding Linux installations, and what you can do to keep these subjects under control. It is available on the Web at:

`http://oss.software.ibm.com/developer/opensource/linux/papers.php`

A very useful paper, it discusses the most common tools and utilities for increasing the level of security, and refers you to various URLs for further information.

Another excellent paper entitled "Linux Security State of the Union", by Robb Romans and Emily Ratliff, is also available at the same URL. The authors discuss the main prejudices regarding open source software in general and Linux in particular, which often inhibit the use of these products in production environments. The contention is that an open source operating system need not be insecure. On the contrary, the availability of the source code availability implies the advantage of a very stable product (because everyone is able to run and test it), whose bugs are fixed with extraordinary speed.

The paper lists and describes projects that are underway to improve overall acceptance of Linux as a secure operating system, ready for productive usage in an enterprise environment.

5.3 Cryptography on z/Series

S/390 and zSeries servers offer specialized processors for cryptographic operations. The IBM 2064 zSeries 900 supports up to eight PCI-CC cards. Each PCI-CC card contains two engines and is assigned two CHPID numbers. Using these cards for cryptographic operations, the CPs or IFLs are released from these processor-absorbing instructions.

In an ISP or ASP environment, cryptographic procedures are generally used for secure TCP/IP connections between the server and the user somewhere in the Internet. Applications for firewalls, Web serving and mail serving have the requirement to protect data, as shown in Figure 5-1.

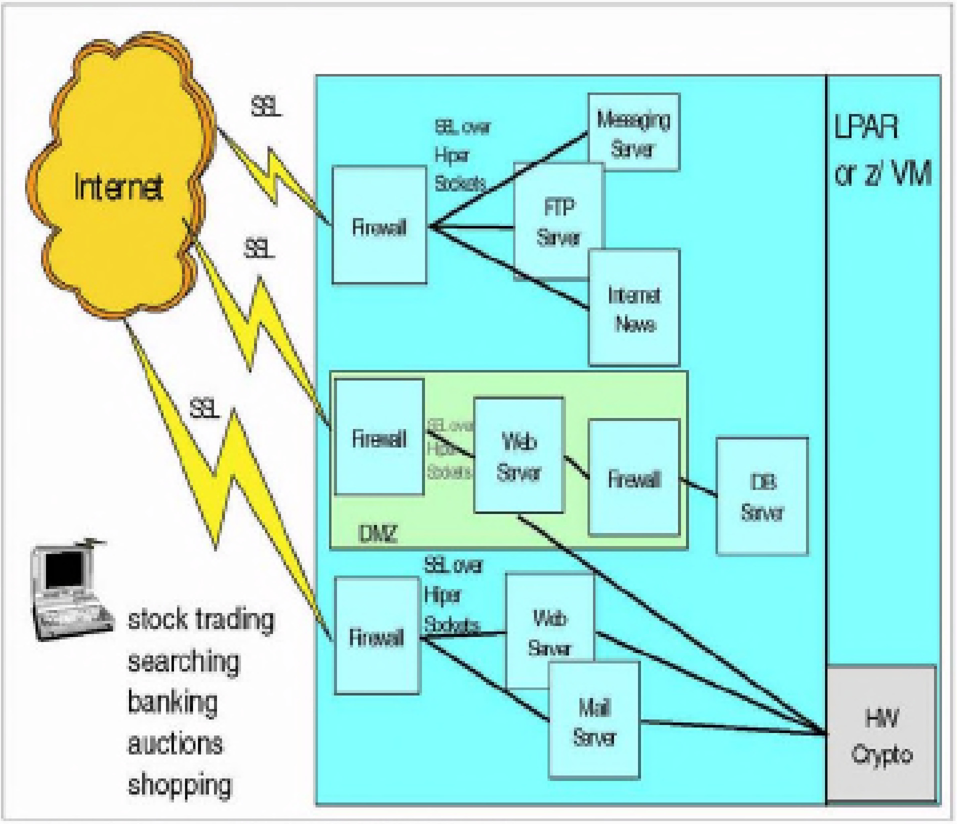


Figure 5-1 Usage of zSeries 900 PCI-CC card

The operating system has to be able to recognize cryptographic instructions, pass them to the PCI-CC card to be executed, and return the result to the application.

Linux on S/390 and zSeries will provide this functionality for hardware encryption in the first step for those asymmetric algorithms used by SSL, which will result in remarkable performance enhancements for SSL transactions. The future directions will also include hardware cryptography for symmetric algorithms (such as DES, 3DES, RC4, etc.), support for cryptographic file system, and digital signatures.

From a security point of view, there are no system integrity exposures if the PCI-CC card is shared and used both with OS/390 and Linux workload. Aside from potential performance concerns if there are not enough crypto features, there are no security considerations. Each operation is discrete and independent of those that precede or follow it. VM manages the queue of requests to ensure that a guest can see only its own request, as with a shared processor or a shared channel. Of course, if necessary, a PCI-CC processor can also be dedicated to a Linux guest system in a virtual machine.

6

Migration planning

In this chapter we discuss high-level considerations for planning a migration from distributed servers to Linux on S/390 or zSeries.

6.1 Where to start

The reality of any large organization's IT infrastructure is that it defies easy analysis, so how do you select which servers are the best candidates for migration? By solving the biggest problem first; focus on the area where there is the most "pain", and fix it.

6.2 What to look for

Consider the following when selecting which servers to migrate.

6.2.1 Small scope

The most successful very early migration customers we've been involved with were faced with these small-scope requirements:

- Needing to upgrade multiple servers running on older hardware
- Needing to upgrade multiple servers running on older operating systems
- Needing to increase disk capacity
- Being unable to justify underutilized single servers

Normal attrition presents many low-risk opportunities to consolidate single-use servers. This builds a track record of success and customer satisfaction. The financial comparison is made using real data.

This type of migration will not justify the purchase of a new S/390 or zSeries; instead, it assumes there is extra capacity on an already installed machine. However, the incremental cost of running the migrated servers can be very accurately measured in terms of the sunk costs in the data center. When compared to the costs of many upgrades to many small servers, this incremental approach is very attractive financially, especially in terms of technical support costs. File and print servers are the typical examples of this migration, and have been very successful.

Medium scope

Other customers have taken a more comprehensive approach. Rather than wait for servers to reach end-of-life before migrating them, a more global and proactive plan is used. Some indicators for using this approach are:

- Consolidating a new acquisition
- Building, expanding, or moving to a new data center
- Implementing a plan for increased capacity
- Redesigning or upgrading a network

This type of migration depends on having strong skills already in place in the S/390 datacenter. It assumes there is plenty of capacity on the machine—or a budget to buy more capacity. It further assumes end-to-end cooperation from all members of the technical support staff. The people who will support applications on Linux need to see this as an opportunity to enhance their skills and marketability. That applies equally to the NT administrator, the MVS system programmer, and the VM system programmer.

Large scope

Another type of migration happens in companies that perceive a strategic advantage in exploiting open source code. These companies chose to bet their business on this model and do not want their competitors to be aware of it, so there are few references. This approach can take the form of a new, "from the ground up" deployment, or be a corporate-wide directive to embrace Linux and migrate to it. The success of such an implementation depends on:

- Strategic, core-business application deployed on Linux
- Strong development, testing, and support staff
- Executive sponsorship
- Competitive advantage

6.3 Total cost comparison framework

In this section we discuss the components you should use to calculate the total cost comparison.

6.3.1 Staffing

There is a current and growing shortage of skilled IT technicians. Leveraging the effectiveness of employees you currently have, and attracting the skill set coming out of college, is a significant financial advantage. Some components of staffing costs are compared in Table 6-1:

Table 6-1 Staffing cost components

Penguin colony	**Separate servers**
One location	Many locations
One staff	Much staff, or downtime for travel
Centralized skill redundancy; provides overlap and backfill	Isolated experts; expensive, hard to duplicate for 24x7 coverage
Mass, automated customization	Manual customization

6.3.2 Hardware

Hardware price comparisons of single-use servers to a mainframe might seem to be an easy win for the single server. A production environment includes more than just a box. When the cost of hardware for failover, backup, peak capacity, and networking is included, the mainframe can become very competitive. A calculation of hardware costs should include these items:

- New servers: CPUs, keyboards, monitors
- Disk storage
- Networking hardware: switches, routers, racks and cabling
- Uplift for failover, backup, and peak capacity

6.3.3 Occupancy

Data centers represent large sunk costs, and server consolidation on Linux zSeries presents a huge opportunity to get the maximum return on that investment. Using the Linux Community Development System example, 600 servers exist in the floor space of a refrigerator. The cost per square foot for single-use servers, no matter how efficiently stacked, is going to be a significant financial burden. Occupancy costs include:

- Floor space
- Heating and cooling
- Electrical consumption
- Uninterrupted power supply

6.3.4 Other factors

The costs of staffing, hardware, software, and occupancy can be measured and projected with some accuracy. Those costs do not represent the whole picture, and it is important to consider the following:

- Cost of an outage
- Cost of migration
- Cost of exploiting emerging technology
- Cost of multiple databases
- Cost of multiple copies of one database
- Cost of a proprietary architecture
  - Non-portable code
  - Restricted interfaces
  - Removal of function

It may not be possible to put a cost figure on each of these, but one of them may be the determining factor that makes Linux on a mainframe the best choice in a particular situation.

7

Backup and restore

In this chapter we discuss how to effectively back up and restore Linux systems. We describe a number of backup methods, and focus on the use of VM and Linux tools.

7.1 Backup methodologies

Backup and restoration, particularly in large environments, involves more than simply making a copy of data and keeping it in a safe location (although that is often a good start). Instead, each different type of data loss scenario requires a different approach to the restoration of service.

7.1.1 Disaster recovery

In order to recover from a disaster such as loss of a computing center or disk subsystem, you must consider hardware considerations as well as data management considerations. Once you solve the hardware issues, you need to make all of your data available again as quickly as possible.

Taking device-level backups is the easiest way to do this. However, this tends to be the most disruptive and costly backup method. As an example, Point-to-Point Remote Copy (PPRC) is an example of device-level backup that is extremely effective in a disaster recovery role, but it requires that you duplicate your entire disk storage facility, which may be cost-prohibitive (this is discussed in more detail in 7.2.2, "Point-to-Point Remote Copy (PPRC)" on page 131).

Using VM DASD Dump Restore (DDR), another method of device-level backup, requires that you shut down the system using the disk being backed-up, which is also disruptive (as discussed in 7.3.3, "VM DASD Dump Restore (DDR)" on page 138).

7.1.2 Logical backup

It is often not appropriate to restore entire file systems at a time. For example, if a single file is accidentally deleted from a server, the most efficient method of getting that file back would not be to have to restore an entire file system.

Device-level backup processes are not appropriate for this purpose. Consider PPRC in this scenario: if the file is deleted on the main disk, within an instant the deletion is repeated on the mirrored disk.

The same thinking applies in the case of data corruption. An external backup solution must also be available so that the file can be restored to a time before the data corruption took place.

7.1.3 Backup types

There are generally considered to be two types of backup: *full* and *incremental*.

A full backup is a copy of the entire file system. This backup could be used to restore that file system if it was completely destroyed. In a device backup, this would be an image copy of the entire device. In a logical backup, every file in the file system would be copied.

An incremental backup only copies *changes from the time of the last backup*. On its own, an incremental backup could not restore an entire file system. Incremental backups are used as a way to keep full backups up-to-date, without having to repeat the entire full backup every time. Incremental backups are usually associated with logical backup methods, where directory information provides an easy way to find out which parts of the file system changed since the last backup.

7.1.4 Complex application backup

Applications such as databases, which generally keep very large files with complex internal structures, pose problems for backup processes due to this internal complexity. A backup program, looking at the file system level, only sees a large file, and must handle it as such. Even if only a single 1 KB record in a database file of 1 GB has been changed since the last backup, an incremental backup of the file system would still back up the entire 1 GB file.

One approach to avoiding this is to use application-specific tools to back up the application data internally. That way, the file system backup facility can be told to ignore the application files. For example, there are a number of tools for DB2 that function within DB2 to perform backups.

Another approach gives the file system backup tool the intelligence to look inside the application file format and treat it almost like another file system type to be backed up. The TSM agent for Lotus Domino is an example of this, allowing TSM to view documents inside .NSF databases and back them up individually.

7.1.5 In-service backup

One method of providing backup for application-specific data is to use a client of that application to back it up. This can reduce the overhead of running a backup client in addition to the application server. We refer to this strategy as “in-service” backup, since the backup happens within the service being provided.

For example, if you have a number of virtual servers providing DNS, the DNS zone data can be backed up by having another DNS server elsewhere in the environment configured as a secondary DNS. This secondary DNS will perform zone transfers to maintain its copy of the configuration, thereby creating a backup of the data. A similar approach could be taken with HTTP or FTP servers, using site mirroring programs such as `wget`.

In many cases, the overhead of a backup client on an application server will not be significant. However, from a security perspective, a highly secure server will have the minimum number of services running in order to reduce security exposure. In this case, an in-service backup strategy may be a suitable option.

7.2 Hardware possibilities

You have the following hardware possibilities for backup and restore processes.

7.2.1 FlashCopy

FlashCopy is a feature of the IBM Enterprise Storage Server that can create an identical copy of disk volumes without interrupting operating system access to the volume.

Note: Other storage systems have similar capabilities, often called by different names. Refer to the documentation for your hardware to check whether your storage systems support such a feature, or refer to your vendor.

This feature can assist the backup process in environments where it is not convenient to take systems down or to take applications offline. Using FlashCopy, data can be duplicated to alternate disk volumes, and then brought online to another system and backed up using traditional methods (tape, etc).

FlashCopy can provide the same support to Linux systems, but it is important to consider the impact of the buffer cache. If a FlashCopy is performed while there are buffered writes in cache, data integrity is lost. In a Linux environment, the following steps would have to be taken:

1. Suspend updates to applications and databases.
2. Flush the Linux buffer cache with the **sync** command.
3. If VM minidisk caching is used, ensure it is flushed also.
4. Initiate the FlashCopy.
5. When the copy is done, reopen applications and databases.

FlashCopy is designed to copy entire disks at once. This means that copying a single minidisk will generally involve copying the entire disk on which the minidisk resides.

7.2.2 Point-to-Point Remote Copy (PPRC)

Point-to-Point Remote Copy (PPRC) is a feature of S/390 and zSeries disk controller hardware that allows a remote disk unit to maintain a synchronized copy of the data held on a production disk. The remote disk unit does not have to be in the same location as the production unit (but distance restrictions do apply).

PPRC is used extensively in “traditional” S/390 installations to provide disaster recovery for entire disk enclosures. Since the data is synchronized at the time of write I/O, the remote disk is virtually identical to the local disk. If the local disk subsystem is lost, the channel paths to the remote disk can be brought online and processing resumes at the point of failure.

PPRC incurs a slight overhead in I/O duration, due to its synchronous nature. The operating system does not see the I/O complete until after the remote unit has successfully completed.

In a Linux scenario, PPRC can be used as part of a highly redundant and available permanent storage design. An example of this design is shown in Figure 7-1.

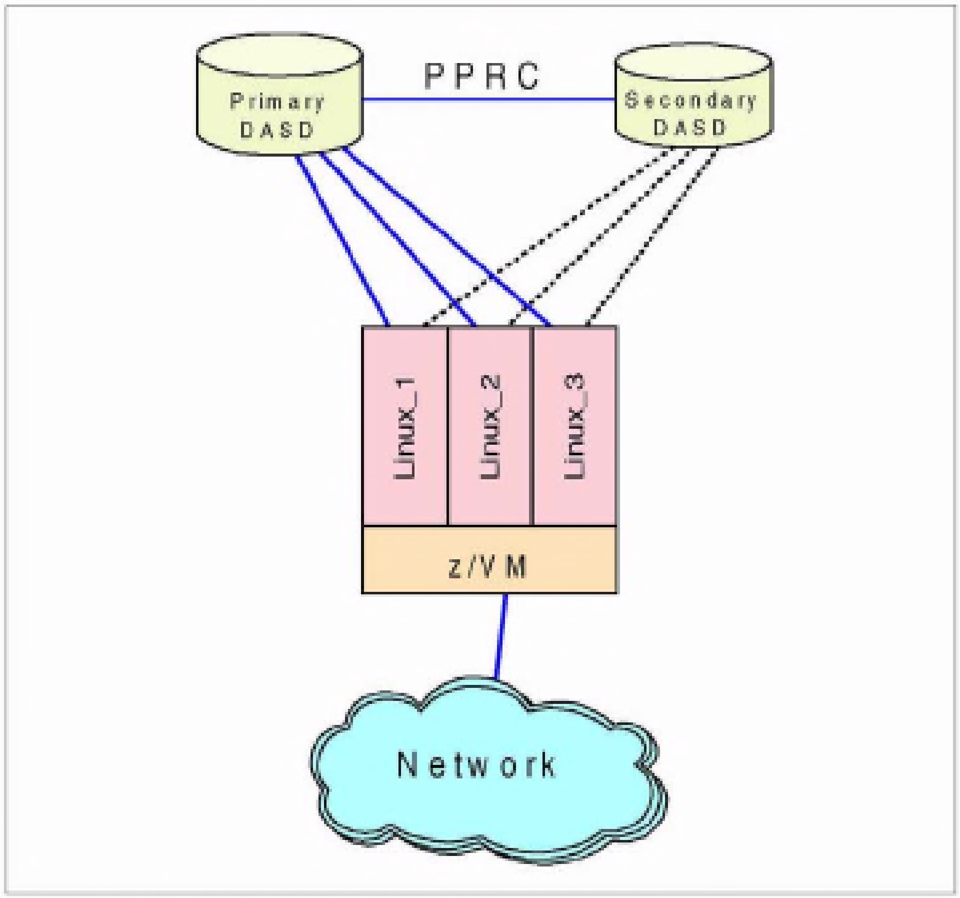


Figure 7-1 PPRC disk configuration for Linux

In this diagram, PPRC is used to synchronously mirror writes to a second disk subsystem. In the event of a failure of the disk, the paths to the secondary disk unit are brought online and processing can resume. However, while VM is able to handle these device switching non-disruptively in many cases, the Linux guest systems and the applications running in them should be shut down if the primary DASD is lost, and restarted with using the secondary DASD.

This design is enhanced and discussed further in 7.4, "High availability choices with zSeries" on page 139.

Note: In "Network block device" on page 137, we discuss a way to produce a similar mirroring method in Linux, which does not require PPRC.

7.2.3 IBM 3480/3490 tapes

IBM has written a driver for IBM Model 3480/3490 tape drives. This driver allows a mainframe tape device to be used by Linux applications and utilities. As with other device support code, it can be either compiled into the kernel or built as a module (tape390.o).

Important: The tape390 driver was written after the SuSE GA distribution was released, so in order to use the driver on this distribution, you need either to get an update, or to build the module (and a new kernel) using an updated source tree.

We used a beta of SuSE Enterprise Server for S/390 7.2, dated June 21 2001, which was built on a 2.2.19 kernel and did contain the tape390.o module.

Configuration

The driver takes a single parameter, which allows you to specify the devices to be used by the driver. For example, the following command would load the tape390 module, defining any tape devices between device addresses 0180 and 0183 and one at 0189:

```
insmod tape390 tape=0180-0183,0189
```

If you want to have the module loaded automatically when it is required, add the following lines to /etc/modules.conf:

```
alias char-major-254 tape390
alias block-major-254 tape390
options tape390 <module-options>   # if you want to pass options to it
```

The specifics quoted here seem to be common in the documentation *Device Drivers and Installation Commands*, for both the 2.2.16 and 2.4 kernels.

> The driver may search for all tape devices attached to the LINUX machine, or it may be given a list of device addresses to use. If it is not given a list the numbers allocated are volatile - the number allocated to any particular physical device may change if the system is rebooted or the device driver is reloaded. In particular a device brought online during a LINUX session will be allocated the next available number at the time it comes online, but at the next reboot it will be given a number according to the sequence of device addresses. If a `tape=` parameter is present at system startup or module load, all tape devices in the ranges of the specified parameter list will be used. The devices are then numbered (sequentially from zero) according to the order in which their subchannel numbers appear in the list.
>
> In both cases the associations between subchannel numbers and device numbers are listed in the file /proc/tapedevices.

Operation

The driver interfaces with programs via /dev nodes, with the same name format expected by standard Linux tape utilities:

- Character mode:
  - /dev/ntibm*n* (non-rewinding)
  - /dev/rtibm*n* (rewinding)
- Block mode:
  - /dev/btibm*n* (for read-only)

Currently, a device major node number has not been formally allocated to this driver, so major node number 254 is being used until a formal allocation is made. To use the tape device driver, you must create the /dev nodes manually (unless your distribution has already created them).

The minor numbers are allocated in pairs starting from 0, with the even number representing the rewinding device and the odd number for the non-rewinding. The even number is used for the block device. The driver allocates drive number 0 to the first device found, 1 to the second, and so on.

For example, to create the /dev nodes for the first two tape drives present in the system, the following commands are used:

```
# mknod /dev/rtibm0 c 254 0
# mknod /dev/ntibm0 c 254 1
# mknod /dev/btibm0 b 254 0
# mknod /dev/rtibm1 c 254 2
# mknod /dev/ntibm1 c 254 3
# mknod /dev/btibm1 b 254 2
```

When a major number is formally allocated and the driver modified accordingly, all that will be needed is for the /dev nodes to be recreated (and the /etc/modules.conf entries, if present, to be edited) using the right major number. No changes to the programs that read or write tapes will be necessary.

Important: If you use devfs on your system, the entries in the /dev tree will be managed automatically, and will have a format derived from the device address of the tape drive being used.

For example, the rewinding device for the tape drive on device address 0181 will be referred to as: /dev/tape/0181/char/rewinding.

For more information on the tape driver, refer to *Linux for S/390 Device Drivers and Installation Commands*.

7.3 Software tools

You have the following software tool possibilities for backup and restore processes.

7.3.1 Software RAID

Using kernel support in Linux, multiple disk devices can be assembled into a disk array using Redundant Array of Inexpensive Disks (I/ORAID) principles. RAID provides many options for aggregating physical devices into contiguous storage, simultaneously providing fault-tolerance and a degree of disaster recovery capability.

Note: RAID is usually implemented in hardware, with specialized controller devices doing the disk aggregation and presenting a single disk volume to the operating system. For this reason, we have to refer to software RAID as a special case.

On S/390, Linux can utilize software RAID by creating a single RAID volume across disk devices on separate disk controllers.

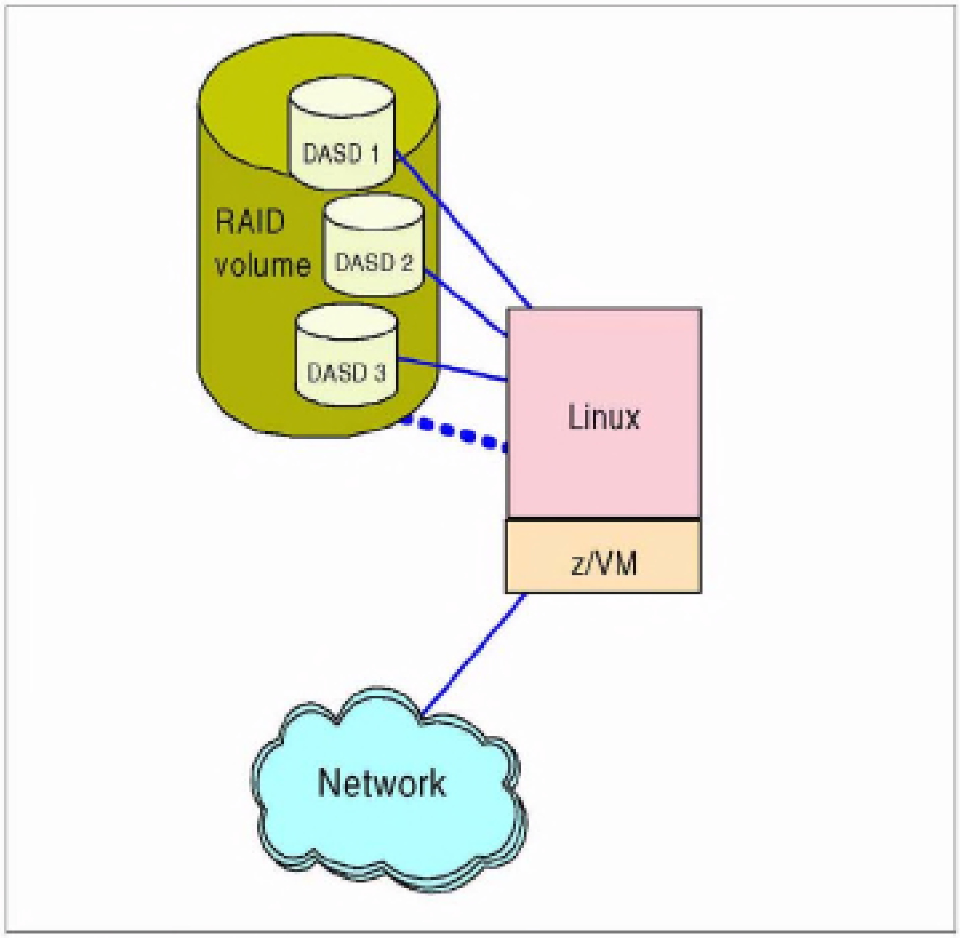


Figure 7-2 Software RAID example

In Figure 7-2, the solid lines represent the real I/O path to the physical devices involved, and the dashed line shows the path to the RAID volume created. Since the file system is created on the logical volume, file system I/O appears to be to the logical device. The RAID driver maps this to an operation to the physical disks.

The greatest benefit in an S/390 environment is gained when the physical DASD is distributed across multiple controllers. In this case, depending upon the RAID configuration used, the logical volume can continue to be available if physical disk is lost.

7.3.2 Network block device

The network block device is a Linux device driver which can provide access to a physical disk across a TCP/IP network. On its own, it does not offer much to Linux on S/390, but combined with a mirroring RAID configuration, it can create an off-site duplicate of a disk volume in a similar way to what PPRC does at a hardware level.

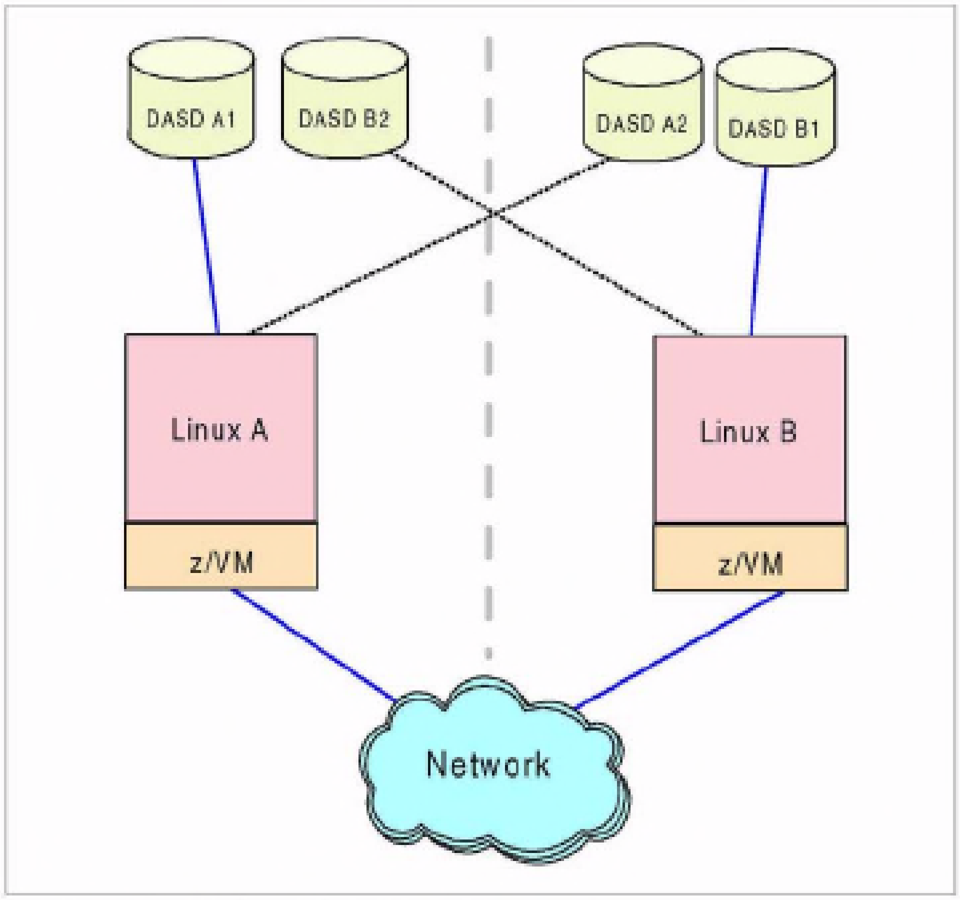


Figure 7-3 Network block device cross-site

In the example shown in Figure 7.3, Linux systems operate on two separate processors in two sites. Disk devices at both locations are accessed only by the Linux systems local to the site. Some of the disk, however, is linked via the network block device to the Linux system in the other site, creating the connectivity shown in the diagram (where logically, each system has access to disk physically at the local site, and via the network to the other site). By creating a mirroring-RAID volume over these devices, a cross-site mirrored device is created.

The network block device can be used in conjunction with distributed file systems (see "Global File System (GFS)" on page 51 for a brief description of how GFS can utilize the network block device).

Using this kind of configuration would provide a cheaper alternative to PPRC for sites that do not currently have the infrastructure to support PPRC. However, to maintain disk response times, high speed network connectivity is needed between the two locations. In certain high-demand applications, the cost of providing the support infrastructure for PPRC might be justified.

7.3.3 VM DASD Dump Restore (DDR)

DDR is a full-volume disk backup utility that can copy VM minidisks in their entirety. These backups do not provide any awareness of the contents of the volume being copied, they simply treat the volume to be copied as a disk file. DDR can be used as a means of providing full-volume backup and restore capability to Linux systems running under VM. Using DDR, entire systems can be dumped to tape, disks at a time.

Important: This operation is best performed when the Linux system is shut down. If you take a DDR copy of a Linux minidisk, any pending writes that Linux has in buffer cache will not be present on the backup. This will lead to data integrity problems. Also, since the Linux file system is mounted when the copy takes place, if a restoration is required a file system check will take place when it is first mounted. For ext2 file systems in particular, this may be undesirable.

DDR can be used to take a backup of a Linux system immediately after installation. In the event of a disaster, this backup could be restored and used as a starter system for subsequent file-level restoration using another tool.

7.3.4 Amanda

Amanda is an acronym for the "Advanced Maryland Automatic Network Disk Archiver"; an Open Source backup scheduler. Amanda uses a client-server arrangement to facilitate the backup of network-attached servers. Using Amanda, it is possible to have a single tape-equipped server back up an entire network of servers and desktops. We provide more detail about its use in Chapter 12, "Backup using Amanda" on page 269.

7.3.5 Tivoli Storage Manager (TSM)

TSM provides data-level, multiplatform, consolidated backup capabilities. TSM agents exist for a variety of server and desktop operating systems, including z/OS, z/VM, Linux for zSeries and S/390, Windows, and others. TSM backs up remote file systems over the network to a central point which copies the data to tape.

TSM offers a number of benefits:

- On supported platforms, TSM provides agents to allow data-level backup of file structures like DB2, Lotus Domino, and others. This means that TSM can back up individual documents within a Notes NSF database, for example.

> **Note:** At this time, Linux for zSeries and S/390 is *not* one of the platforms supported for TSM agents, so TSM can only provide file-level backups.

- Integration with hierarchical storage management facilities, such as those included with DFDSS on z/OS.
- A single backup platform and scheduling point for the entire data center.

TSM is also described in the IBM Redbook *Linux for zSeries and S/390: Distributions*, SG24-6264, which is available on the Web at:

`http://www.ibm.com/redbooks/abstracts/sg246264.html`

7.4 High availability choices with zSeries

In the preceding sections, we describe the procedures to backup and restore VM and Linux data. But whether whole minidisks are backed up using DDR, or incremental copies of single Linux datasets are produced by TSM or Amanda, these backup copies are all out of date the moment after they have been created. And if data has to be restored by using these backup copies, then updates that were issued to the data—after the last backup copy was produced—are usually lost.

In cases where data is destroyed in a logical way (e.g. by some erroneous program code), then going back to an old but safe copy of the data is probably acceptable. But what happens if all data belonging to several Linux guests—or even to the entire VM LPAR—is destroyed?

In such a case, a disaster recovery effort is required. As pointed out previously, restoring data from backup copies created by DDR and TSM is very time-consuming. New hardware has to be provided, installed and defined; the data from the backup copies has to be restored; and changes to the data after the last backup copy was made will still be missing. Similar problems arise if the server itself is damaged in a disaster.

Traditional architectures provide a reliable environment

With many Linux servers under VM, the traditional S/390 and zSeries configuration for setting up high availability computing can be exploited to provide a reliable Linux operating environment.

S/390 and zSeries architecture is designed for continuous availability, which means that the services have to provide both high availability (the avoidance of unscheduled outages) as well as continuous operations (the avoidance of scheduled outages). In an OS/390 or z/OS environment, this goal is usually reached by exploiting the design of a Parallel Sysplex, enriched by remotely mirroring DASD devices to build a Geographical Dispersed Parallel Sysplex (GDPS).

While Linux is not able to participate in a Parallel Sysplex, all other architectural design points a GDPS is based on can be used to design a highly available Linux production environment on zSeries. This includes the use of remote DASD mirroring to provide consistent copies of all data, the use of Capacity BackUp (CBU) to enable required computing power at the surviving server, and the use of automation to handle all procedures for bringing up the backup systems and redefining network and channels.

Figure 7-4 on page 141 shows a sample scenario, which we discuss in detail.

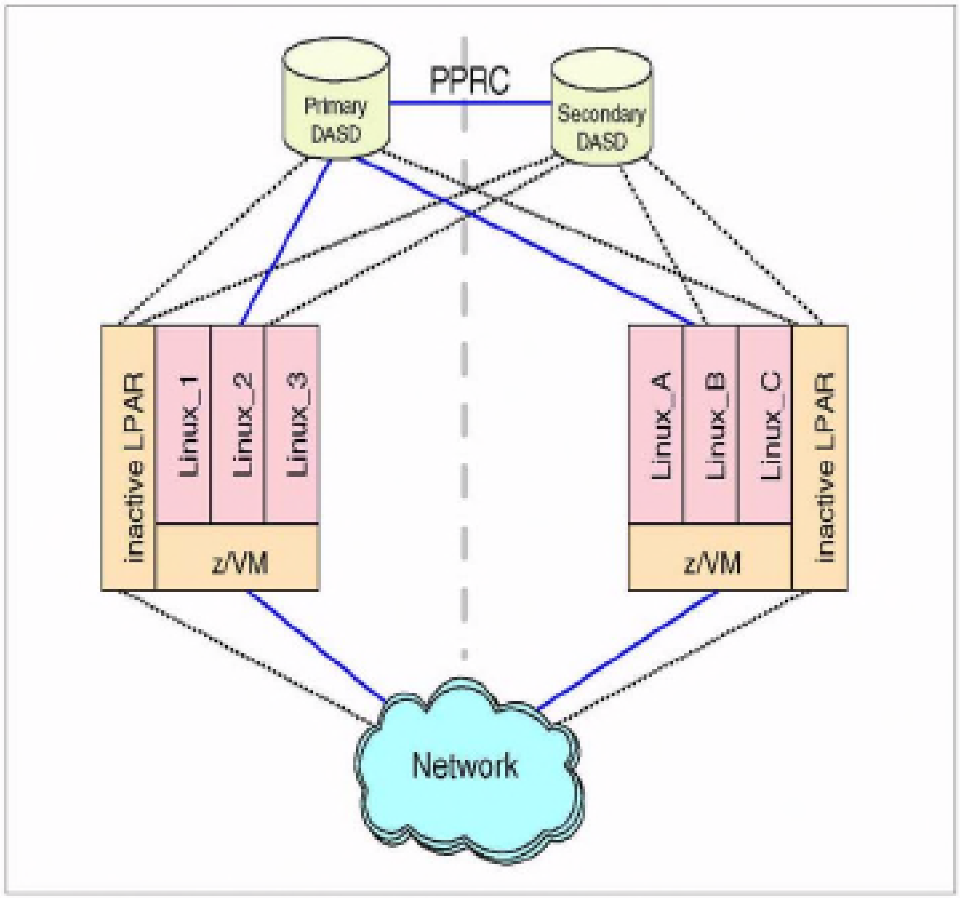


Figure 7-4 High availability Linux scenario, normal operations

In this example, the Linux servers operate in two z/VM LPARs on two zSeries servers, which are located in two physically separated computing centers (indicated by the dotted grey line between the servers). Each of these zSeries servers also contains a defined, but inactive, LPAR, which is supposed to take over the workload of the other server in case of a disaster.

In order to offer sufficient computing power to this LPAR without hampering the work of the Linux guest systems in the other productive LPAR, additional redundant processors are available that can be concurrently activated by using the Capacity BackUp (CBU) feature. CBU is available on 9672 and zSeries z900 servers; refer to the IBM Redbook *IBM @server zSeries 900 Technical Guide*, SG24-5975, for more details. It is available on the Web at:

`http://www.ibm.com/redbooks/abstracts/sg245975.html`

The data belonging to the z/VM operating systems of both zSeries servers, as well as the minidisks of all the Linux virtual machines, are located on a DASD control unit (CU) attached to both servers, using FIbre channel CONnections (FICON) or Enterprise System CONections (ESCON). The DASD devices are remotely mirrored to another DASD CU, by using the synchronous Peer-to-Peer Remote Copy (PPRC) abilities, provided, for example, by the 2105 Enterprise System Server (ESS).

The operating systems (z/VM and Linux) are *not* aware of this mirroring; only the connections to the primary DASD CU are activated. Nevertheless, connections to the secondary DASD CU are defined and cabled (indicated by dotted lines), ready to be used in the case of a disaster.

Also, the network connections are defined to both LPARs in each zSeries server, but only the connections to the productive LPAR are active.

7.4.1 Loss of a DASD control unit

The following scenario discusses what happens if the primary DASD control unit (CU) fails, as illustrated in Figure 7-5 on page 143.

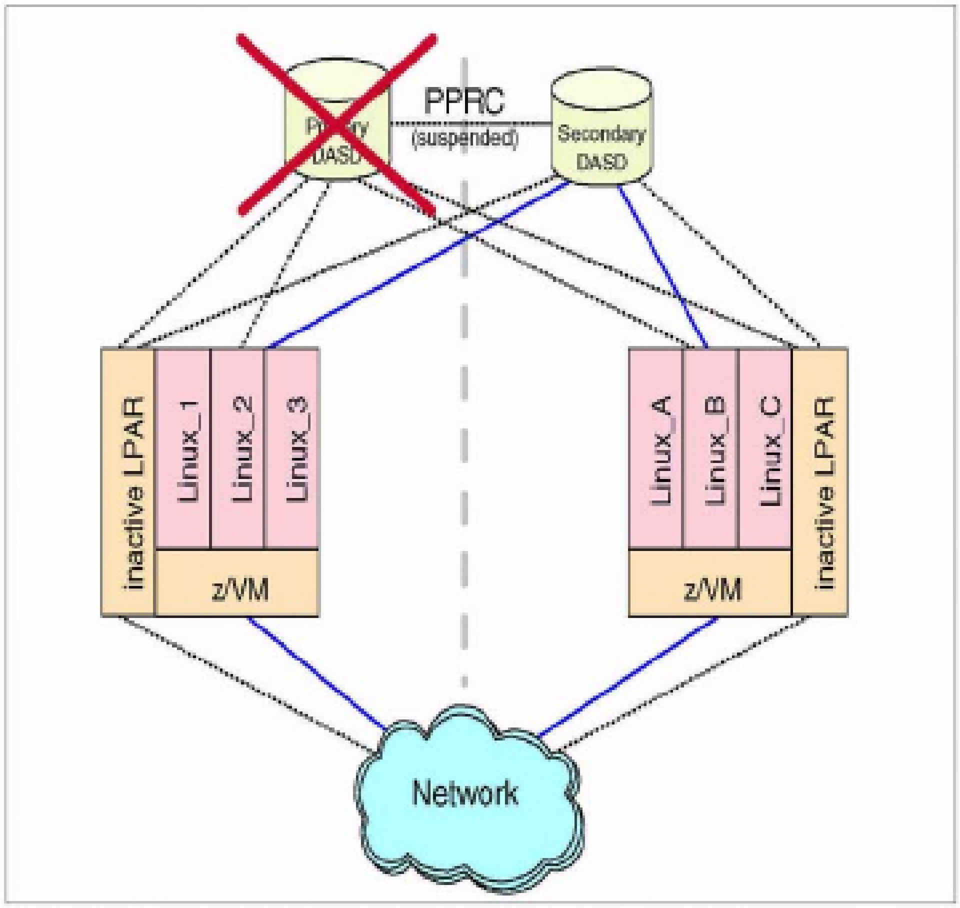


Figure 7-5 High availability Linux scenario, after failure of DASD control unit

If the primary DASD CU becomes unavailable for some reason, the mirrored data on the secondary CU has to be used. In order for this to happen, the devices of the primary CU have to be configured offline to all attached operating systems, and the devices on the secondary CU have to be configured online.

Because the secondary devices are simply mirrored images of the primary ones, they contain the same data and even have the same VOLSER, and the failed I/O operations of the attached systems can be set up again on the secondary DASD hardware.

Depending upon how fast the switch is performed, and on the time the applications and operating systems accept a delay in I/O operations, the applications may continue to work without being disrupted. However, with Linux running as a VM guest operating system, all Linux images affected by the failing primary DASD should be shut down and rebooted from the secondary DASD.

It may be possible, depending upon how often the Linux system accesses the minidisks on the failing DASD, that the secondary DASD can be configured online by CP, without Linux having noticed the temporary absence of the minidisks—or it can at least be possible to repair these minidisks with Linux still running. But this will require manual intervention in the Linux system to ensure that the restored minidisks are working correctly and the data is not corrupted.

Therefore, for ease of use and to ensure data integrity, in the case of the loss of one or more DASD volumes, we recommend the following:

- Configure the failing devices offline to VM.
- Shut down the affected Linux systems.
- Configure the secondary devices online.
- Restart the Linux systems with using the secondary devices.

With the loss of the primary DASD CU, mirroring with PPRC is of course suspended, and further operations have to continue without mirroring until the CU is available again. The failure of the secondary DASD CU, the failure of the PPRC connection between the both CUs, and the loss of the access to one or several devices also need to be considered.

For example, if the zSeries server is able to issue a write I/O operation to the primary DASD device, but the primary CU is not able to operate mirroring to the secondary CU because the connection between the CUs units has failed, the I/O by default will not be completed. In this case you have to decide if the I/O operations to the primary CU should be resumed without mirroring, or if the operations have to be stopped.

The primary DASD CU does not know why there is no response from the secondary CU; possibly the entire second computing center has been destroyed, or it may only be the result of a weak cabling connection. For this reason it will probably be necessary to get more information regarding the state of the secondary CU before deciding how to continue with the I/O operation.

The procedures to gather the necessary information, the rules for making the appropriate decisions, and the execution of the required commands have to be coded and established by using automation utilities. Besides the basic hardware functionality, this is one of the most complex topics of a GDPS.

7.4.2 Loss of a S/390 or zSeries server

The next scenario involves the failure of a whole zSeries server. If a disaster strikes the computing center, and the server is not able to continue operations, the workload has to be transferred to the surviving server in the other computing center. This is illustrated in Figure 7-6 on page 145.

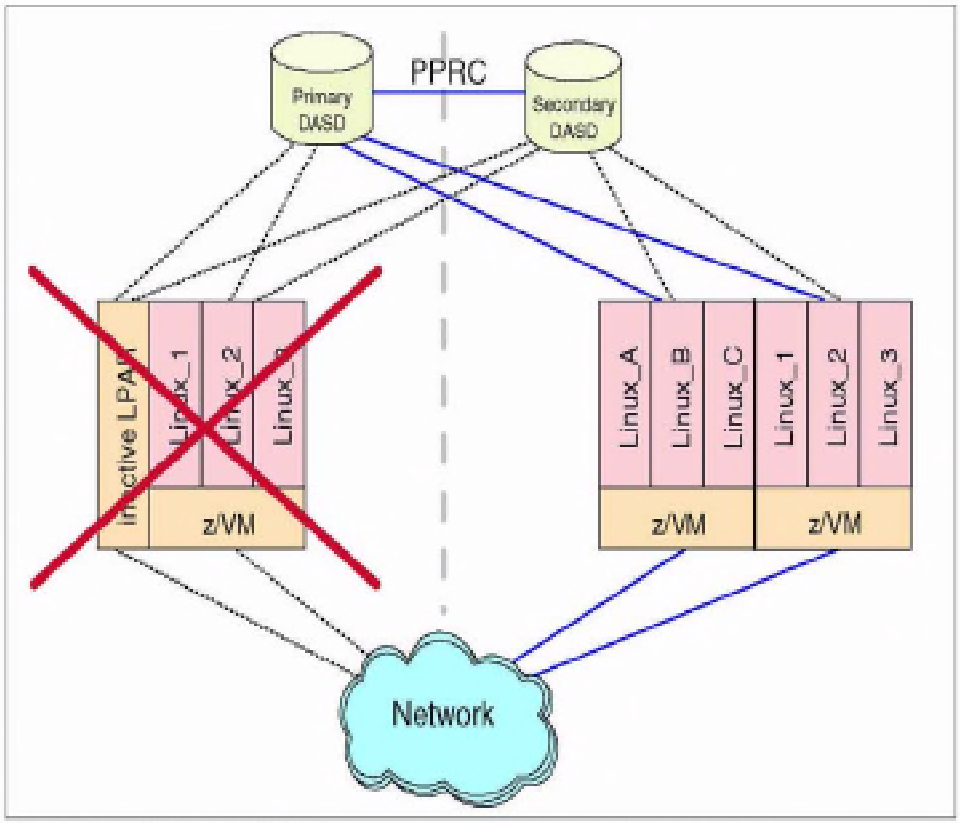


Figure 7-6 High availability Linux scenario, after failure of server system

On the surviving server, an additional LPAR has already been defined for disaster recovery purposes. This LPAR is now activated by loading the operating system from the same DASD devices as the failing system used, using the already installed connections to the primary DASD CU. The PPRC mirroring operations between the primary and secondary DASD CU continue without any changes.

While the LPAR activation on the server can take place immediately after the failure of the original server, the processor capacity of the surviving server has to be adjusted as soon as possible, in order to meet the needs of the combined workloads of both servers. This can be done by exploiting the 9672 or 2064 z900 CBU feature, which allows you to activate physically available but unused processors non-disruptively.

Resuming network connections to the restored LPAR is also no problem, since the LPAR activation is done with exactly the same data as the failed original LPAR, and the same network addresses as before are used. The OSA of the new LPAR on the surviving server makes itself known to the network with these addresses, and network operations resume as before; the fact that the physical hardware has changed is transparent to the routers in the network.

In summary, the hardware capabilities of PPRC and CBU allow you to achieve a high availability computing environment based on Linux under VM on S/390 or zSeries. But the scenarios we presented require a thorough understanding of the reasons for failures, the actions that have to be taken to recover from failures, and the consequences of these actions. Although it is possible to handle such recovery tasks manually, as computing environments become increasingly complex, there is a greater need for automated processes.

8

Performance analysis

In this chapter we discuss the performance analysis and tuning of Linux images running under VM.

Performance of a large system is critical to its success, only one step away from functionality. Once a system has been developed and is functional, the next question is—what is the price performance? If price performance is very low, a system and service is more likely to succeed. The intent of this chapter is to help you realize an optimal price performance.

8.1 Performance considerations

The aspects of performance measurement and tuning that are needed in environments serving many users are extensive. This chapter reviews server resource analysis, subsystem analysis, server delay analysis and some level of response time analysis. The methods of storing performance data is also a consideration; is report format sufficient, or is a performance database necessary to allow detailed analysis—and even accounting functions—to be performed? In an environment where service is associated to charge-back, the performance database becomes more important.

S/390 and zSeries systems are large systems with many potential points of contention, and many places of potential optimization. Each subsystem can be complex and have many options. Any subsystem, when overutilized, can be a global system bottleneck. As your workload changes or grows, utilization of one resource can reach a threshold that causes a significant *global* resource shortage. On a system where changes on one server can impact performance of another, you need to monitor the environment carefully.

Many early Linux for S/390 installations did not have VM performance skills—and ran into problems which easily could've been avoided. For example, the most common complaints often were "things stop" or "this system is slow". Both of these issues can result from very simple configuration problems.

VM has a very sophisticated scheduling mechanism that is designed to efficiently support thousands of users, even when resources are overcommitted. Linux has some features that impact the view of resource utilization. Understanding storage and page space requirements when designing a system allows you to avoid configuration and utilization problems.

8.2 Why measure performance

Performance measurement and tuning has a cost in terms of staffing and time. In environments where the cost of purchasing a new system is less than the cost of analyzing the performance of a system, it is common to not spend much time on performance analysis. However, in a zSeries and S/390 environment, most installations will require both real time analysis as well as a structured methodology for capacity planning. This section describes methodologies for both real time performance analysis and capacity planning.

Projecting requirements when moving applications from an NT or Linux/Intel environment to zSeries and S/390 is important, in order to ensure a high level of success. If the resource requirements of an application exceed what is available or what makes economic sense, then that needs to be understood prior to moving the application to zSeries and S/390. Choosing applications to run effectively on zSeries and S/390 will greatly increase the chance of success.

8.2.1 Cost of running applications

In environments such as ISP and ASP, where charge-back is reasonable, the cost of running each application is a consideration. Each application should be evaluated to determine which platform is most cost effective. Some applications will be more cost effective on dedicated RISC processors, while others will be more cost effective on zSeries and S/390.

Applications that require resource for only short periods of time are very effective in an S/390 time-sharing environment, because when the applications are idle, they do not consume processor or storage resource. As the applications become active, they are brought back into memory. The cost of storing the application on disk (page space) can be measured in cents-per-megabyte and is usually insignificant.

However, applications that require significant dedicated resource have a very different cost model—instead of using an amount of storage for a small percentage of time, these applications require storage most of the time. For these applications, there is no choice other than to ensure the resource is provided all the time. Because the cost of dedicating resources is much higher, the cost of running this application is also higher and potentially should be put on a platform suitable to the requirements. This may be an LPAR, or a non-S/390 server; the platform decision should be based on what is economically justified.

8.2.2 Controlling costs

Once the cost of running an application is known, you'll want to monitor those costs. If an application or server suddenly increases its resource requirements, either due to programming errors or increased workload, the cost of running that application will rise. Given a charge-back environment, there will be issues if the additional costs are not recognized as they occur; sudden increases in monthly costs can cause problems. By monitoring costs as they are incurred, both customers and service providers will have a better understanding of the provided service.

To ensure the cost of running an application does not increase without management awareness, you will need to have a performance monitoring methodology. This methodology should include thresholds for resource use and should be monitored programmatically. Mechanisms for alerting management of unexpected increases in resource requirements are required components of managing this type of service.

8.2.3 Controlling the impact of one application on another

In environments where there may be several different customers and where Service Level Agreements are required, there's a need to monitor each application and server on a real time basis to ensure that problems with one customer do not impact other customers. It is common practice to provide an automated alert system to check operational values against threshold settings. Details on implementing this function are discussed in 8.13, "Alerts" on page 175.

8.3 Measurement tools

Following is a list of some of the measurement tools available on VM:

- ESALPS from Velocity Software

  See 8.3.1, "Measurement tool used" on page 151.

- FCON/ESA from IBM

  FCON/ESA provides performance monitoring capabilities with system console operation in full screen mode. FCON/ESA can give you an immediate view of system performance, or post-process its own history files. Threshold monitoring and user loop detection is also provided, as well as the ability to monitor remote systems. The most recent enhancements include support for remote access via APPC, virtual disk in storage reports, and enhanced minidisk cache reports.

- Real Time Monitor (RTM) VM/ESA from IBM

  With RTM VM/ESA, you can get an immediate view of current system performance. Use RTM VM/ESA for short-term monitoring, analysis and problem-solving. It can simplify performance analysis and the installation management of VM/ESA environments. The latest RTM VM/ESA service includes support for the RAMAC array family, and support for RTM VM/ESA to run in 370 Accommodation mode on VM/ESA Version 2.

- VM Performance Analysis Facility (VMPAF) from IBM

Use VMPAF to statistically correlate and chart system performance problems, tuning information, and trend analysis. VMPAF does this by analyzing the relationships between variables from VMMAP, VMPRF, monitor and other data sources in order to determine which subsystems are most impacting your current system performance. Then, using interactive graphics, VMPAF gives you a quick, clear picture of these relationships.

- VM Performance Reporting Facility (VMPRF) from IBM

  VMPRF uses your system monitor data to analyze system performance, and to detect and diagnose performance problems. VMPRF provides interim reports, plus summary and trend data that show resource use by user ID or user class, response time, DASD activity, channel utilization and system throughput.

8.3.1 Measurement tool used

The measurements in this chapter were performed using the Linux Performance Suite (ESALPS), a commercial product from Velocity Software that is available on the Web at:

http://VelocitySoftware.com

The ESALPS components we used were: ESATCP, for gathering Linux and Network performance data; ESAMON, for processing the VM performance monitor records and data gathered by ESATCP; ESAMAP, to provide reports for long-term analysis. ESAMON and ESAMAP are based on the CP monitor providing current performance data and long-term historical data. (ESAWEB, the fourth component of ESALPS and a VM-based Webserver, was not used for these measurements.)

ESAMON creates a Performance Data Base (PDB) to store performance data. This data is then used as input for reporting in many forms. There are two forms of the PDB, referred to in the ESALPS documentation as "History files". The first form has a one-minute granularity and allows a detailed analysis. The second form has a 15 minute granularity and is long term, with each day's data being summarized.

In this chapter, we provide examples and give recommendations about how to use these effectively. The reports and real time displays provide the performance information in the same format. Each report and display has a name (such as ESAUSRC) which provides user configuration data. Menus and tables of content help users find the needed reports and displays. Performance reporting is performed for:

- User data, showing resource requirements by user, user class, accounting codes.

- Response time data, showing response times based on the CP definition of transactions.
- Processor subsystem, showing details of all processors and LPARs.
- DASD and DASD Cache, showing DASD response times, by I/O component, cache controller data showing cache hit information, read/write data, etc. MDC hits (I/O satisfied by the CP minidisk cache function) by device are provided, as well as MDC hits by user. Data is provided both by device and by control unit. Channels are measured, and seek analysis is provided.
- Storage subsystem showing user storage, MDC storage, CP storage and available storage.
- Paging/Spooling subsystems, showing device and system activity, as well as utilization.
- Non-DASD I/O showing tapes, network controllers, channel-to-channel adapters and any other attached device, showing both by device and by control unit.
- TCP/IP data showing traffic at each layer of the IP stack (Transport layer, IP Layer, Interface/Hardware layer), and for the local VM stack, traffic and network response times by subnet and by application.
- Linux data showing by resource utilization by application (processor and storage), disk utilization, storage use for cache and buffer. Data is provided for any Linux being monitored.

8.3.2 Screen display

Screens can be displayed by from a CMS virtual machine executing “ESAMON screen”. The examples in this chapter can be displayed in this manner. For example, issuing the command **ESAMON SMART** gives you the screen shown in Example 8-1. This screen is automatically updated every 60 seconds.

Example 8-1 Output of ESAMON SMART command

```
TUNER: SMART   ITSO                                         08/09 13:51-13:52
 Seconds until next interval: 57                                   2064 40ECB
   <----------Top Users----------> <-----------Servers----------->
   Userid:     CPU% IO/Sec Pg/Sec Userid:     CPU% IO/Sec Pg/Sec
 1) ESAWRITE    0.33   0.80      0 VMLINUX9     1.8  27.67   0.02
 2) VMRTM       0.23      0      0 VMLINUX7     1.7   0.75      0
 3) ESASERVE    0.00      0      0 VMLINUX2     1.1   0.45      0
 4) ESATCP      0.00      0      0 TUX8MSTR     0.3   0.37      0
 5) HUB6        0.05      0      0 VMLINUX6     0.3   0.02      0
 6) SNMPD       0.00      0      0 TUX60002     0.2      0      0

<---------CPU Statistics-----> <---In Queue User statistics----> <-Page->
%cpu %usr %prb %sys %ovr %idl   InQ   Q0   Q1   Q2   Q3 Eli Ldng <-rate->
```

```
8.86 7.48 5.97 1.39 1.50 190.   17.3 1.98 1.08 0.28 14   0    0    2.3
```

8.4 Measurement data sources

With environments including Linux servers and VM hosts all linked in a network, there is a need to measure each environment to understand the full system. However, each environment of the system has different measurement requirements and data sources, so combining the different data sources to understand your capacity and performance will be the challenge. Following is a list of the common data sources for each environment.

- Network - the most common method of evaluating network performance is by using a Simple Network Management Protocol (SNMP) data source.This is well defined and includes many network performance metrics. It has been extended by many vendors and open source groups to include information found useful for specific network components. SNMP Version 2C is the most common. This includes password protection of the data (community names), and performance enhancements in the protocol.
- z/VM - measuring VM is typically done using the CP Monitor as a data source. This technology scales very well and is widely used. The monitor reports on almost everything that has been thought to be useful on reporting subsystem and user performance for VM systems. As new releases of z/VM come out, and new releases of TCP/IP appear, the monitor is enhanced to provide instrumentation for new features.

  Most performance analysis products suitable for a large environment will be based on the CP monitor. An alternative to using the CP monitor is to use a diagnose interface to look at internal VM control blocks. This provides access to most data. The downside is that every release of VM requires the data interface to be updated, whereas the CP Monitor is automatically updated with each data source, allowing users of the CP monitor to run without impact on new releases of VM.
- Linux - how to measure Linux from a global perspective is new technology. A good open source performance data source is NETSNMP. Linux is being enhanced significantly and the associated data source must be enhanced as well. NETSNMP provides performance data accessible to network monitors using SNMP. This performance data includes the network traffic, a set of private MIBS from University of California/Davis (UCD MIBS), and HOST MIBS that are defined in RFC 1157. The HOST MIBS provide data on applications for processor and storage requirements, as well as data on each

device of the Linux system. There is active development of NETSNMP, with new releases regularly.

NETSNMP is included in the three major Linux distributions (Red Hat, SuSE, Turbolinux) providing a common data source for analyzing performance. It can be found on sourceforge at:

http://net-snmp.sourceforge.net/

See 13.6.2, "SNMP installation" on page 306 for more detailed information on installing NETSNMP.

- CP Monitor - there are many options as to what data to have the monitor produce. For real-time monitoring, an interval of 60 seconds is common, balancing the overhead of collecting the data with the problem of a very long interval that hides performance spikes.

  Measuring Linux wait states seems to be much more useful with a state sampling of .1 (10 times per second). The overhead of this method seems to be immeasurable, but it's a reasonable place to start. If you are using ESALPS, then all domains should be enabled except for SEEK and SCHEDULE (these should be enabled when you are performing specific analysis and require SEEK or SCHEDULE data).

8.5 Local vs. global performance tuning

In an environment with many virtual servers, having the technology and personnel to optimize the environment is a necessity. Optimizing a large, single, expensive system has significant payback, while optimizing small minicomputers often takes only a few minutes (or consists only of paying for inexpensive upgrades). However, tuning a large system that shares resources entails different requirements, ones that may be surprising to installations that are new to the zSeries and S/390 environment.

Virtual machines (Linux servers) should be tailored to the applications. Allocating resources to one virtual machine takes resources away from other applications—the difference between thinking globally and thinking locally. Minimizing resource requirements for one application means more resources are available for other applications; this reduces your overall costs.

Recognizing there are performance people from two radically different environments (local and global), tuning and performance analysis must be designed for the appropriate environment.

8.5.1 Local environment

Local environments use dedicated servers and tend to have one application on a server; they then tune that server to meet the application requirements. For some applications (Web serving, Lotus Notes, print serving), multiple servers might be dedicated to one application. The benefit of this approach is that work occurring inside one server does not impact work on other servers. Because the cost of each server is small, the cost of incremental growth is not necessarily a capital expenditure.

8.5.2 Global environment

The traditional performance methodology is two to three decades old and is based on global optimization of a system, evaluating systems with many applications sharing resources. In this global type of environment, resources are typically more expensive and therefore sharing those resources is a primary objective.

When one application or user on the system consumes large amounts of a resource, it impacts other applications or users on the system. Thus, global optimization requires you to evaluate all aspects of performance—from both a resource subsystem perspective and from an application perspective. Most large installations have dedicated personnel just for performance analysis, capacity planning and system optimization.

With current directions, and with cost justifications for moving many smaller servers to fewer and larger zSeries and S/390 systems, the optimization perspectives must be global—one server in a virtual machine can impact other servers on that system.

8.6 Linux operational choice

There are two methodologies for operating Linux; probably the most efficient is to use a mix of both. One is the typical VM/CMS methodology, where one virtual machine runs one (and only one) application. The other methodology is more like a typical VM/VSE or centralized Linux server environment, where one server runs many applications. The advantage of running small servers with only one application is that the server can be tuned for the application. The advantage of the larger server running many applications is a reduction in overall overhead.

Security considerations also influence the operational choice; having many applications running on only one server increases the risk an application falling prey to a hacker allowing access to data from multiple applications. Using single application servers greatly reduces your security risks.

See Chapter 5, "Security architecture" on page 105 for a discussion on security.

8.7 The CP scheduler

The multiprogramming level of z/VM is probably higher than that of any other platform. With possibly tens of thousands of users, each running their own programs and environment, there is a requirement for sophisticated task management and scheduling. The CP scheduler provides this function. The scheduler determines when users run, how often they run, which users to restrain when a resource is constrained, and many other functions. There are several controls that installations can use to tune the scheduler. Generally, the default values for these controls are appropriate.

Note that the scheduler has been tuned to run thousands of CMS (single tasking) users, as well as 10 to 20 large multitasking operating systems such as VSE, TPF or OS/390. Some installations may run even 100 larger guests. The operational considerations of running thousands of Linux servers on the z/VM system are not completely known. Education and new methods of tuning will likely be required.

8.7.1 Queue definitions

The scheduler categorizes each user by how long it has been running. Short tasks fall into queue 1, intermediate tasks are in queue 2, and long-running tasks are in queue 3. There is also a queue 0 for high priority work.

Linux servers that run a timer to do some small amount of work every 10 mS break this model. With the timer interrupt every 10 mS, CP classifies any Linux server as a long-running task, and will put it in queue 3. An implementation of the timer routines in Linux without using the 10 mS interrupt has been proposed, but is not yet available in the mainstream kernel sources.

However, measurements of a preliminary implementation showed the expected reduction of CPU resource usage for idle Linux servers. The measurements also showed that CP again was able to distinguish transactions, and did not classify every Linux server as a queue 3 user all the time.

When the Linux server is dispatched less frequently, you will have more control; long-running jobs are likely more resource-intensive, and you can reduce the number of queue 3 servers allowed to compete for resource. Reducing the concurrent number of tasks competing for resource then reduces the contention felt by the shorter tasks.

8.7.2 Global controls

The two most useful SRM controls are the DSPBUF and LDUBUF.

DSPBUF is used to control the number of users accessing the processor. Generally, the DSPBUF is not needed; however, when your processor is constrained, you can use the DSPBUF to limit the number of queue 3 users allowed to compete for the processor resource, which will in turn reduce the processor utilization. Thus, even in a very processor-constrained environment, short tasks will run very fast.

LDUBUF is used to control the number of users allowed to page. The default value of LDUBUF allows a system to thrash (the point where pages are being paged out for one working server to allow another working server to resume work). If you get to this point, the only solution is to reduce the number of servers competing for the paging resource. Lowering the LDUBUF from its default value does that. Linux servers have working sets that are variable and typically very large. Because of this, the scheduler may not react as fast as you'd like to current storage demands, so you may need to use STORBUF and XSTOR to achieve the desired effect.

STORBUF limits the amount of storage in use by dispatchable users. The STORBUF control is more often a hindrance than an assist to performance. Most performance people recommend raising the STORBUF control in order to make it non-operational. The XSTOR operand tells the scheduler to treat some percent of expanded storage as main storage (the usual recommendation is 50% for this value).

Virtual machines that are being held back due to resource constraint are kept on a list called the "eligible list". If you never have any users on the eligible list, the scheduler is not detecting a shortage of resources that could be alleviated by holding some users back. Thus, if LDUBUF is holding users back, then users would otherwise be loading in working sets which the paging subsystem may not be able to support, the DSPBUF reduces the number of dispatchable users, and the STORBUF limits the amount of storage in use by dispatchable users.

Following are the shortcut recommendations for scheduler controls. The numbers following the set command are for: a) all queues, b) queues 2 and 3, and c) queue 3. This allows you to set the amount of contention for each resource by queue.

Note: You should raise the queue 3 value of DSPBUF if processor utilization never exceeds 90%, and lower it if processor utilization often is at 100% for long periods of time.

```
SET SRM STORBUF 300 250 200
SET SRM DSPBUF 32000 32000 30
SET SRM LDUBUF 80 60 40
SET SRM XSTOR 50%
```

One method for measuring the impact of the scheduler on users is to look at the ESAUSRQ display, which provides most of the needed information. Knowing when there are users that are being held back by the scheduler because of the settings for LDUBUF or DSPBUF tells you which resource is constrained.

8.7.3 Local controls

There is a local control for each server that should be used sparingly. Setting a virtual machine to QUICKDSP tells the scheduler to never hold this user back, regardless of resource constraints. Virtual machines such as your TCP/IP gateways, security monitors, and the operator, should have this option.

Use QUICKDSP only to protect specific virtual machines when resources are very constrained. Using it too often will disable one of the most important features of the scheduler: the ability to survive serious resource shortages.

Priority between virtual machines is provided by the use of SHARE settings. There are two options for SHARE settings, relative and absolute. Users with a *relative* share will get a smaller overall share as more users logon. Users with an *absolute* share maintain their share regardless of the number of other virtual machines.

Thus, users such as TCP/IP or security managers should be given absolute shares as their workload increases when more users logon. Relative shares are used to control how the remaining resource is divided up between the rest of the virtual machines. The following recommendations can be given.

- Set the shares to absolute for all service machines that you expect will need to provide more work as more virtual machines are created, and set the shares to relative for all others.
- Set the ABSOLUTE shares to the peak required value (for example, 8% for TCP/IP if that is TCP/IP's requirement at peak load).
- Using the default of RELATIVE 100 is recommended unless you have a need to prioritize work.

Do not use very high relative shares, because using high relative shares for one or more users reduces your ability to prioritize your production work. For example, if there are 10 servers, and all servers are relative 100, then each server is scheduled to obtain about 10% of the system.

The arithmetic is quite simple: the "Normalized Share" is the relative share divided by the total of the relative shares. If one of the servers is then given a relative share of 200, that user gets a significant increase of about 9% (from 100/1100 to 200/1100).

Giving one user (TCP/IP, for example) a relative share of 10,000 means that each default user has a share of 100/11000, or less than 1%. There is no need to confuse the scheduler by reserving 90% of the resources for a guest that only needs 5%. Proper tuning of your system means allocating absolute shares to your key service machines.

A second part of the local controls is setting a cap on how much processing power a user is allowed to absorb. There are two reasons to do this: for accounting (to ensure users do not get more than what is paid for), and when users loop, or run very CPU-intensive jobs (to minimize their impact on other servers).

The following sets a target average share, but limits the server to 5% of the processing resource. The effects of the LIMITHARD are measurable on the ESAXACT (transaction analysis) report.

```
SET SHARE REL 100 ABS 5% LIMITHARD
```

8.8 Processor subsystem

Knowing how much processor is used, and by which servers, is information you need to know for efficient capacity planning. Controlling the rate at which your servers access the processor is done by setting Shares. Share settings have a minimum value and a maximum value, with several options and variations of each.

Using the CP monitor, you can capture over 99.9% of the processing power used. Building a processor map showing how much processor is used by LPAR, VM, Linux servers, VM servers, and CMS allows you to project future requirements based on new users or customers.

One of the issues seen in the past was the following: an important Linux application was ported in a fashion guaranteed to produce poor performance. As the Processor Local Dispatch Vector Report (ESAPLDV) in Example 8-2 shows, there were about 70,000 dispatches per second on each processor—this should be about 1000 on systems that are running well. This overhead was very costly; running anything 211,000 times per second would intuitively have a very high cost!

This is the kind of potential problem that's extremely hard to diagnose without the proper tools. An installation might perceive that S/390 performance is bad—when, in reality, a simple correction to the application might eliminate 200,000 calls to a function that does not need to be called.

On a dedicated processor, this might not be an issue. However, on a zSeries or S/390 system where most resources are shared, this application would be inappropriate to run as it performs in this example.

Other items to examine in the report are the number of steals and moves per second (low numbers are desirable). The `Moves To Master` value indicates how many calls are made to functions that must be single-threaded on the master processor. High numbers indicate use of functions that may not be appropriate for a high performance application.

The `PLDV Lengths` values show the number of virtual machines waiting on each processor queue, indicating the current level of multiprogramming. This is a good example of the need for VM performance analysis *and* Linux performance analysis.

Example 8-2 Sample Processor Local Dispatch Vector Report (ESAPLDV)

```
<VMDBK Moves/sec>  <--------PLDV Lengths-------> Dispatcher
CPU  Steals  To Master   Avg  Max Mstr MstrMax %Empty Long Paths
 -   ------  ---------  ----  --- ---- ------- ------ ----------
 0    823.8        0.7   0.2  1.0    .       .   83.3    70489.8
 1    111.1          0   0.4  1.0    .       .   55.0    70454.2
 2    196.4          0   0.4  2.0    .       .   61.7    70056.8
     ------  ---------  ----  --- ---- ------- ------ ----------
     1131.3        0.7   1.0  4.0    .       .  200.0   211000.7
```

8.9 Storage subsystem

Lack of storage to meet the requirement results in paging, and paging causes delays in service. Monitoring the storage requirements and the impacts of applications provides necessary feedback for capacity planning. There are many ways to reduce storage, and on VM there are different types of storage. For storage capacity planning purposes, you should maintain a map of your storage to understand the requirements for VM, Minidisk Cache, Linux user storage (by customer), VM Servers (TCP/IP, management service machines), and CMS users, if any. This map should be maintained for both Expanded Storage and Real Storage.

8.9.1 Storage options

For storage (memory), there are several options, each with different impacts on Linux, applications, and your global resources. Coming from a minicomputer or microcomputer environment, administrators have been taught that swapping is undesirable. When swapping to slow SCSI devices, this may be true, but on zSeries and S/390, there are many other options—and these options can reduce your overall (global) resource requirements. For swapping, the alternative options are:

- Use Virtual disk as a swap device. The benefit is a much smaller page space requirement, as well as a smaller requirement for real storage.
- Use RAMdisk as a swap device. The benefit is a smaller requirement for real storage. When sharing storage between many servers, this is important.

8.10 DASD subsystem

For DASD (disk) storage, there are options to share some amount of disk between servers, read only. Using VM's minidisk cache to cache shared data once is significantly more effective than having each Linux cache the same data.

There are currently[1] three different ways to format the disks to be used by Linux.

dasdfmt	The DASD driver in Linux for zSeries and S/390 comes with the `dasdfmt` utility to format the disks. It formats all tracks on the disk with a fixed block size. There is no support for this particular format in existing S/390 software.
CMS FORMAT	The FORMAT program in CMS also formats the disk with fixed block size, but adds a special eye catcher in R3. This format is recognized by CMS and by CP.
RESERVE	With the CMS RESERVE command, a single big file is created to fill the entire (CMS-formatted) minidisk. The Linux file system is then built into this single big file such that the original CMS formatting of the disk is retained.

There is a small penalty for using the CMS RESERVE format in that some of the blocks on the disk are not available for use by Linux. These blocks are used for CMS housekeeping, as shown in Figure 8-1.

[1] The patches that were made available on June 29, 2001 appear to change several things in this area. We have not yet investigated what the impact of these changes is.

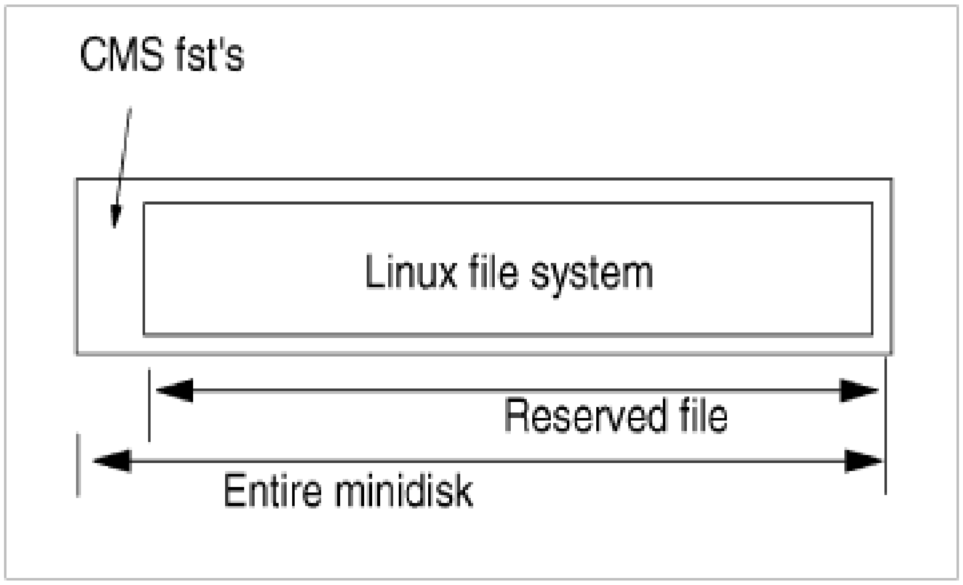


Figure 8-1 Minidisk prepared with RESERVE

However, the advantage of this approach is that the disk can be accessed by a CMS user ID and is clearly identified as in-use to everyone. CMS applications can even read and write the blocks in the file (for example, with the **diskupdate** stage in *CMS Pipelines*). An extra bonus may be the fact that the big file on the disk has a file name and file type which gives you 16 more characters to guide systems management processes (like writing with a marker on CD-R disks that you created).

Linux can also use a disk that was only formatted by CMS. In this case Linux will use all the blocks on the disk such that CMS ACCESS will fail on this disk afterwards. Even when you do not need the ability to access the blocks from CMS, there may still be a good reason to prefer this format over Linux **dasdfmt**. VM directory management products like DirMaint can format the disk before making it available to the user ID.

Tip: If you use IBM RAMAC Virtual Array (RVA), there would be a benefit if you use the "Instant format" function. The "Instant format" is part of the VM SnapShot function such that an instant copy of a formatted (empty) disk is made on the extent to be formatted using the SnapShot feature of RVA. This copy is instantaneous and does not occupy back-end storage in the RVA.

We believe there is no good reason to use the **dasdfmt** command for Linux images on VM, except for the situation where you have a virtual machine running Linux and you forgot to format the disks. Since Linux does not yet tolerate DETACH and LINK of minidisks very well, you'd have no option otherwise but to shut down the Linux system and get back to CMS to format it (but if you do an automatic format with DirMaint, that would not happen anyway).

Note: There used to be a bug in the DASD driver that prevented Linux from booting from a CMS RESERVEd minidisk. This resulted in the recommendation to avoid that format when you wanted to boot from disk.

This is bug was fixed long ago. You can make a CMS RESERVEd mini disk bootable with **silo**. Whether you want to do that on VM, or use a NSS to IPL from, is another matter.

8.10.1 VM Diagnose I/O

The biggest advantage of the CMS RESERVE style of format, however, is that it is the only disk format for which the current Linux for S/390 DASD driver can use VM Diagnose I/O. Diagnose I/O is a high-level protocol that allows the user ID to access blocks on its minidisks with less overhead than pure S/390 channel programs with Start Subchannel (SSCH).

To enable VM diagnose I/O in the DASD driver, you must configure the kernel to enable the "Support for DIAG access to CMS formatted Disks" which is not done in the default SuSE kernel. To enable the option, you first need to disable the "Support for VM minidisk (VM only)" configuration option (also known as the old mdisk driver).

Note: The configuration options in the kernel are slightly confusing in that "CMS formatted disk" really means a disk prepared with the RESERVE command. There is no technical reason why this should be like that. When the kernel is configured without the DIAG option, the DASD driver will use SSCH for the I/O. The VM Diagnose interface does not require the disk to be RESERVEd, so as long as it is fixed-block formatted; Diagnose I/O could have been used for both types.

The DASD driver with the May 14, 2001 SuSE distribution appears to be broken. When configured to use the DIAG support, it refused to use the diagnose interface for the disk that was prepared with the CMS RESERVE command. After fixing the dia250() function in dasd_diag.c to return the correct return code, the minidisk was recognized by the driver as such, but Linux then appeared to hang after it started to scan the partition table. Both these problems have been fixed in the 2.2.18 patches from Linux for S/390, but the fix apparently was not ported back to 2.2.16.

Showing the benefits of VM Diagnose I/O

To quantify the effects of VM Mini Disk Cache (MDC) and Diagnose I/O, we did a simple test using the Linux 2.2.18 kernel with the linux-2.2.18-s390 patch. Each Linux image in the test booted with a RAMdisk and then ran a script as shown in Example 8-3. The loop in the script creates a 64 MB file and then reads it four times to allow some of the I/O be satisfied using the cache. The file is large enough to completely flush the buffer cache of the Linux image.

Example 8-3 Sample script for testing diagnose I/O

```
mke2fs /dev/dasda1 -b 4096
mount /dev/dasda1 /mnt

while [ true ]; do
  dd if=/dev/zero of=/mnt/temp bs=1024 count=65536
  cp /mnt/temp /dev/null
  cp /mnt/temp /dev/null
  cp /mnt/temp /dev/null
  cp /mnt/temp /dev/null
  rm /mnt/temp
done
```

We ran a number of Linux images with this script (on a VM system that turned out of be more I/O-constrained than we expected).

Note: When the MDC design was changed to cache full tracks of data rather than just the 4 K formatted CMS minidisks, this caused problems for database applications that do fairly random access to the blocks on their disk, or at least do not follow a track-related reference pattern. The full track cache was then enhanced with the "Record MDC" (sometimes referred to as "Classic MDC"). Since Linux does not have a track-based reference pattern either, it was assumed that Record MDC would make a difference.

Three different ways to format a disk for Linux, and three different styles of MDC, gives nine combinations, but some of these do not need to be measured; see Table 8-1. Only Diagnose I/O is eligible for record MDC. This means that the Linux DASD driver specifying record MDC for the other two styles of formatting disables MDC.

Table 8-1 Impact of MDC on response time

	No MDC	Track MDC	Record MDC
dasdfmt	30.5 s	17.0 s	N/A
FORMAT	30.3 s	17.1 s	N/A
RESERVE	36.3 s	8.8 s	8.9 s

The difference between track and record MDC is very small in this experiment, because the I/O was mainly sequential and involved a relatively small amount of data. With more random access to the data, one should expect record MDC to waste less storage for reading in unwanted data, and thus be more effective.

Table 8-2 Comparison showing the benefits of MDC

Format	MDC	CPU s	I/O	Elapsed times
dasdfmt	no	1.87	3.09	30.5
	track	1.96	2.44	17.0
FORMAT	off	1.90	3.25	30.3
	track	1.41	1.68	17.1
RESERVE	off	2.37	11.3	36.3
	track	1.91	2.65	8.9
	record	1.93	2.28	8.9

The comparison in Table 8-2 on page 165 clearly shows improved response times when using Diagnose I/O combined with MDC. The channel programs used by the DASD driver appear to be "MDC unfriendly" in that they do not exploit MDC very well. We have not been able yet to understand why this is the case. Considering the obvious advantage of Diagnose I/O, it is not very interesting to fix the channel programs used by the DASD driver when running on VM.

8.10.2 DASD MDC measurement

The following ESADSD2 real time screen shot shows a sample measurement over time to help you to understand the effects of track minidisk cache against block minidisk cache. A block-level copy was done on CMS from the LNX013 volume, one at 11:00 with track cache in use, and one at 11:07 with record cache. Using track cache, about 1500 I/Os were issued; using record cache, 18,000 I/Os were issued. Intuitively, this is reasonable with there being 15 blocks per track; using track cache for sequential I/O should greatly reduce the number of physical I/O.

The following shows the activity to the device over time. When evaluating DASD response time, the response time value is usually the most significant; it shows how much time an average I/O takes to the device. When this value is large, then the components of response are evaluated. The components of DASD are evaluated as follows:

Pend time	This is the time for the I/O to be started on the channel; normally less than 1mS.
Disc time	This is the time for the control unit to access the data. This includes rotational delays, seek delays, and processing time inside the control unit. Disc (for disconnect) time is normally less than 2 to 3 mS on cache controllers.
Connect time	This is the time to transfer the data on the channel, normally less than 2 mS for a 4K block of data.
Service time	This is normally the sum of pend time plus disconnect time plus connect time. In some environments, installations may choose to use just disconnect plus connect, but this is not typical.
Queue time	This is the result of many users accessing the same device. If the device is already servicing another user when an I/O is started, the I/O sits in queue. The length of time in queue is queue time. This is the component of response time that, under load, is the most variable and the most serious.

The sample shows that connect time is high when using track cache, and low when using record cache. Record cache moves one 4 K block of data each I/O, and track cache will move up to 15 blocks of data. This example shows the best case for track cache, being a copy of a large file. More typical is random 4 K I/O, in which case reading in a track of cache wastes transfer time and cache space.

When evaluating performance, data has different requirements. If moving data sequentially, then using track cache can be measured.

```
Screen: ESADSD2  ITSO                           ESAMON V3.1  08/07 10:58-11:16
1 of 3  DASD Performance Analysis - Part 1      DEVICE 3ba1          2064 40ECB


Dev      Device %Dev <SSCH/sec-> <-----Response times (ms)--->
Time      No. Serial Type   Busy   avg  peak  Resp  Serv  Pend  Disc  Conn
-------- *--- ------ ------ ---- *---- ----- ----- ----- ----- ----- -----
10:59:00 3BA1 LNX013 3390-9  0.0   0.5   0.5   0.8   0.8   0.5   0.0   0.3
11:00:00 3BA1 LNX013 3390-9  8.6   8.5   8.5  10.2  10.2   1.3   0.0   8.9 <=Track Cache
11:01:00 3BA1 LNX013 3390-9 15.9  17.2  17.2   9.2   9.2   0.2   0.0   9.0
11:07:00 3BA1 LNX013 3390-9 24.5 197.3 197.3   1.2   1.2   0.2   0.0   1.0 <=RecordCache
11:08:00 3BA1 LNX013 3390-9 12.4 102.7 102.7   1.2   1.2   0.2   0.0   1.0
11:15:00 3BA1 LNX013 3390-9  0.1   0.6   0.6   1.0   1.0   0.2   0.0   0.8
11:16:00 3BA1 LNX013 3390-9  0.1   0.2   0.2   3.6   3.6   1.9   0.0   1.7
```

8.10.3 High connect time analysis

DASD performance when running Linux guests is very different. After reviewing the following analysis, an I/O trace was performed. However, as we will see, sometimes "an I/O is not an I/O": Linux using the Start Subchannel I/O driver will chain over 100 CCWs together - with write operations, up to 130 I/O chained together and perceived as one I/O. At 130 times 4 K blocks, that's over 500 K transmitted per I/O! Sometimes, an I/O is not just an I/O.

The graph in Figure 8-2 on page 168 shows the distribution of read-CCWs over the channel programs[2]. Some 30% of the channel programs have just a single read-CCW and another 30% have 32 reads in them! To phrase it differently: more than 70% of the reads come from a channel program that was reading 128 KB at once.

[2] The reads are simply "command-chained" in the channel programs, not using suspend and resume operations like CP does for paging I/O.

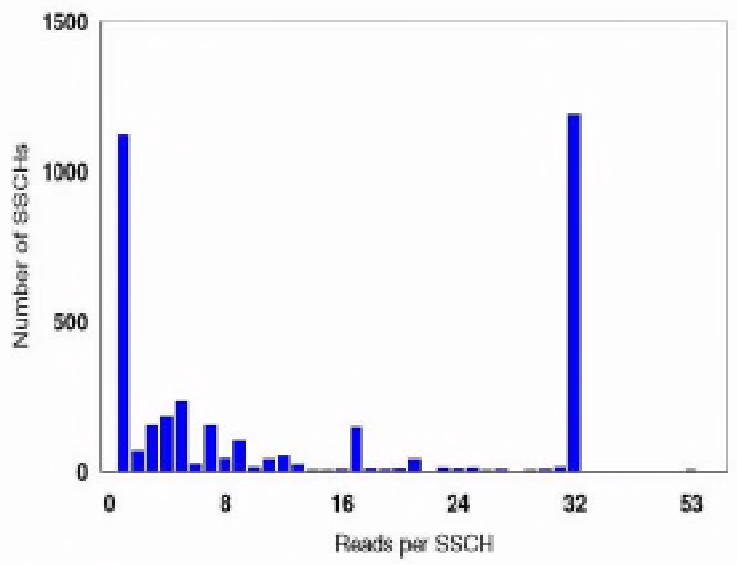


Figure 8-2 Number of read-CCWs in a channel program

The measurements for Figure 8-2 come from a SuSE "default system" install where the contents of the three CDs was copied to a single large minidisk with an ext2 file system. When the CDs were unpacked to the clean disk, very likely files ended up mostly in consecutive blocks in the file system. Because the install process mainly reads large rpm packages from the disk, this makes it easy for Linux to build long channel programs.

While this may not be the typical Linux application we need to run, it is not an artificial situation either because Linux tries very hard to combine multiple disk I/Os in a single operation.

Note: From looking at the read-CCWs combined in a channel program, we get the impression that "track" in the Linux device driver does not match the track on the real device. We did not test this because it would be easier to verify by reading the source code. For efficiency of the control unit cache and MDC, it could be attractive if Linux were more aware of the geometry of the real DASD devices.

Example 8-4 on page 169 looks at a control unit during a test run.The control unit is an RVA. In this case, the operations were 100% read I/O. (High connect times limit the ability of a control unit to service other users, so consideration of the type of work and the appropriate hardware to support the work will be needed.)

In this example, the control unit 3B03, with a range of 256 addresses, is performing a peak of 58.3 I/Os per second, each with an average connect time of 13 milliseconds. Using the 17:21 interval, multiplication of 14 mS times 54 I/O per second shows that a channel is about 75% busy. As channel utilization increases, delays waiting for the channel will occur. You should always ensure that sufficient channel resources are available to meet your workload requirements.

Example 8-4 Control unit during test run

```
Screen: ESADSD2  ITSO                          ESAMON V3.1  08/07 17:09-17:33
1 of 3  DASD Performance Analysis - Part 1     CU 3b03              2064 40ECB


         Dev        Device %Dev <SSCH/sec-> <-----Response times (ms)--->
Time     No. Serial Type   Busy   avg  peak  Resp  Serv  Pend  Disc  Conn
-------- *--- ------ ------ ---- *---- ----- ----- ----- ----- ----- -----
17:17:00 3B03 .     3990    0.0   8.2   8.2   0.5   0.5   0.1   0.0   0.3
17:18:00 3B03 .     3990    0.0  10.7  10.7   7.9   7.9   0.2   0.5   7.2
17:19:00 3B03 .     3990    0.3  49.7  49.7  15.7  15.7   0.9   1.0  13.8
17:20:00 3B03 .     3990    0.4  56.8  56.8  16.6  16.6   0.7   1.0  14.9
17:21:00 3B03 .     3990    0.4  54.2  54.2  16.2  16.2   0.6   1.4  14.1
17:22:00 3B03 .     3990    0.4  58.3  58.3  15.7  15.7   0.8   1.9  13.0
17:23:00 3B03 .     3990    0.2  29.9  29.9  19.2  19.2   0.5   1.0  17.7
```

After looking at the performance analysis of this control unit, the next step is to understand how many paths to the device there are and how they are impacted. Example 8-5 shows there are 4 paths to the devices on this control unit: 41, 4C, 36, and 56.

Example 8-5 DASD configuration display

```
Screen: ESADSD1  ITSO                          ESAMON V3.1  08/07 19:37-19:38
1 of 3  DASD Configuration                     LIMIT 500 DEVICE 3b0 2064 40ECB

 Dev               Device      <----Online CHPIDs----> Ctl Unit UserID    MDisks
 No. SysID Serial Type   Shr 01 02 03 04 05 06 07 08 Model    (if ded)  Linked
---- ----- ------ ------ --- -- -- -- -- -- -- -- -- -------- --------  ------
3BA0 OCFD  LNX012 3390-9 NO  41 4C 36 56 .  .  .  .  3990-3E  .              0
3BA1 OCFE  LNX013 3390-9 NO  41 4C 36 56 .  .  .  .  3990-3E  .             19
3BA2 OCFF  VMLPG2 3390-9 NO  41 4C 36 56 .  .  .  .  3990-3E  .              0
3BA3 OD00  LNX014 3390-9 NO  41 4C 36 56 .  .  .  .  3990-3E  .              5
3BA4 OD01  LNX015 3390-9 NO  41 4C 36 56 .  .  .  .  3990-3E  .              9
```

Now we need to evaluate channel utilization, in order to help understand the performance of this control unit. Evaluation of the channel utilization of the four available channels shows that the multiplication of connect time times I/O rate is a close approximation. Adding up the values of each channel gives a total of 90%. When this approaches 200% (about 8 MB per second), you will likely be able to measure delays associated with channel delay. As it only took one Linux server in this case to push about 3 to 4 MB per second, it might be fairly easy to reach this constraint.

Example 8-6 Channel utilization analysis

```
Screen: ESACHAN  ITSO                           ESAMON V3.1  08/07 17:14-17:32
1 of 1  Channel Utilization Analysis            CHANNEL 00-FF        2064 40ECB

                 <-Pct Utilization->
Time     CHPID     Logical  Physical     Shared
-------- --------  -------  --------     ------
17:21:00    35        0.00      1.67       No
            36        0.00     26.67       No
            37        0.00      3.33       No
            38        0.00      1.67       No
            39        0.00      1.67       No
            3B        0.00      3.33       No
            41        0.00     23.33       Yes
            43        0.00      5.00       Yes
            4A        0.00      5.00       No
            4C        0.00     26.67       Yes
            4E        0.00      3.33       No
            52        0.00      1.67       No
            54        0.00      3.33       No
            56        0.00     13.33       No
```

8.10.4 DASD write analysis

The same analysis was performed for a device with write activity. This device often runs above 60% utilization, which is high by most standards. With a connect time of between 30 to 40 ms, this device would appear to have a severe problem. And an I/O rate of 23.7 I/O per second is not impressive. A CP trace shows this device to be performing write operations that are multitrack of up to 500 KB per I/O.

Example 8-7 Write operations analysis

```
Screen: ESADSD2  ITSO                           ESAMON V3.1  08/07 20:00-20:21
1 of 3  DASD Performance Analysis - Part 1      DEVICE 3ba3          2064 40ECB
```

```
         Dev        Device %Dev <SSCH/sec-> <-----Response times (ms)--->
Time     No.  Serial Type   Busy   avg  peak  Resp  Serv  Pend  Disc  Conn
-------- *--- ------ ------ ---- *---- ----- ----- ----- ----- ----- -----
20:01:00 3BA3 LNX014 3390-9  0.3   0.1   0.1  24.4  24.4   0.3  22.6   1.4
20:08:00 3BA3 LNX014 3390-9  1.0   0.5   0.5  20.1  20.1   0.2  16.1   3.8
20:09:00 3BA3 LNX014 3390-9  0.0   0.2   0.2   0.4   0.4   0.2   0.0   0.2
20:12:00 3BA3 LNX014 3390-9  6.8   1.5   1.5  46.0  46.0   0.5   3.1  42.3
20:13:00 3BA3 LNX014 3390-9 54.4  18.7  18.7  29.0  29.0   1.4   2.1  25.5
20:14:00 3BA3 LNX014 3390-9 84.2  23.7  23.7  35.6  35.6   1.0   2.7  31.9
20:15:00 3BA3 LNX014 3390-9 63.9  18.0  18.0  35.5  35.5   1.3   1.5  32.7
20:16:00 3BA3 LNX014 3390-9 65.5  18.3  18.3  36.8  35.8   1.2   1.5  33.1
20:17:00 3BA3 LNX014 3390-9 32.7   6.8   6.8  47.8  47.8   1.6   0.9  45.3
20:20:00 3BA3 LNX014 3390-9  2.1   1.3   1.3  15.9  15.9   0.5   6.2   9.1
20:21:00 3BA3 LNX014 3390-9  0.2   0.2   0.2  11.6  11.6   0.1   5.1   6.3
```

8.10.5 DASD/cache

I/O response time is made up of several components that include disk rotation time, seek time, data transfer times, control unit overheads and queue time. The technologies to deal with these can be faster disks and different forms of cache (processor-based cache or storage controller-based cache). Example 8-8 further analyzes the data from the previous example. Note that cache is active 100% of the time (Pct. Actv Samp), and three of the four samples were in the 10 to 12% read. This validates the statement that this measurement was of write I/O.

The I/O for write hits on the RVA was almost 100% hit, using DASD fast write. DASD fast write is a function provided by the control unit that accepts the data, and terminates the I/O operation from the host perspective. Then the control unit moves the data to disk. This optimization allows more I/O to be started to the device without waiting for data to actually be written to the relatively slow disks. The small number of read I/O were almost always a "hit", meaning satisfied by data in the cache.

Example 8-8 Cache analysis

```
Screen: ESADSD5  ITSO                            ESAMON V3.1  08/07 20:11-20:15
1 of 3  3990-3 Cache Analysis                    DEVICE 3ba3          2064 40ECB

                        Pct.  <--------------------per second------>
         Dev            Actv  <------Total--------> <----Read---->
Time     No.  Serial Samp   I/O Hits Hit% Read%  I/O Hits Hit%
-------- ---- ------ ----   ---- ---- ---- -----  ---- ---- ----
20:12:00 3BA3 LNX014  100    1.4  1.3 90.8  24.1   0.3  0.3 90.5
20:13:00 3BA3 LNX014  100   18.2 17.6 96.9  12.2   2.2  2.2  100
```

```
20:14:00 3BA3 LNX014  100  23.2 22.3 96.1  10.4  2.4  2.4 97.9
20:15:00 3BA3 LNX014  100  17.0 16.5 97.3  12.3  2.1  2.0 97.7
```

8.11 Network performance

Network performance analysis is part of ESALPS. This allows you to determine what nodes are active, and how much network activity is being generated by each one. The screen in Example 8-9 shows the active nodes. The ones that are recognized as running Linux are noted. The TCPIP and HUB6 are VM TCP/IP stacks.

From this screen, moving the cursor to a node and pressing PF2 will show the configuration of that node. Note that in the Name column, a convention was used to include the virtual machine name in the configuration file when setting up the SNMP daemon on Linux. This allows you to look at this configuration data and recognize which virtual machine is running the server.

Example 8-9 Active nodes sample

```
Screen: ESATCPD  ITSO                           ESAMON V3.1  08/09 13:07-13:08
1 of 1  TCP/IP Node list                        NODE *              2064 40ECB


           Node     IP Address      Name
           ----     --------------- ------------------------

           TCPIP    .               .
           HUB6     .               .
           IVO123   9.12.0.123      nf3000-1
           ITSO237  9.185.246.237   linux7                  (Linux)
           ITSO232  9.185.246.232   linux2                  (Linux)

PF1=Help      PF2=ESATCPC  PF3=Quit  PF4=ESATCPT    PF5=ESAHST4      PA1=CP
PF7=Backward  PF8=Forward                                  PF12=Exit
PA2=Copy
 ====>
```

Performance data comes in different flavors. The ESATCP2 screen shows the IP layer of data. The four screens showing data from the TCP/IP stacks are ESATCP1, ESATCP2, ESATCP3, and ESATCP4.

Looking at a stack as TCP/UDP (transport layer) on top, that is ESATCP1. The next layer of the stack is the IP layer, shown in ESATCP2. ICMP is shown in ESATCP3, and the hardware/Interface layer is shown in ESATCP4.

Example 8-10 shows the IP layer from the test system. In this instance, there was not a lot of activity. What is shown is the number of datagrams forward and delivered. Note that the "HUB" stack is a VM TCP/IP stack in a virtual machine called HUB6, which is acting as a virtual hub between the Linux servers. It forwards all datagrams, rather than delivering them to the local transport layer and applications.

Many errors in TCP/IP are found in the "Discarded" category. There are many reasons to discard datagrams, such as when they are wrongly addressed, or use an invalid port. We suggest you track errors such as these to detect hackers, and applications that have coding errors.

Example 8-10 IP layer from test system

```
Screen: ESATCP2  ITSO                           ESAMON V3.1  08/09 14:14-14:16
1 of 2  TCPIP Internetwork Layer Data           NODE * LIMIT 500     2064 40ECB

                  <Internet Protocol Datagrams per Second > <Datagram output>
                  <Input datagrams> <Discarded Inp Errors >       <Discarded>
Time     Node     Total  Fwrd Dlvrd   Hdr  Addr  Port Other Reqst NoRte Other
-------- -------- ----- ----- ----- ----- ----- ----- ----- ----- ----- -----
14:16:00 TUX8MSTR  1.55  0.00  0.97  0.00  0.00  0.00  0.00  1.32  0.00  0.00
         ITSO232   0.98  0.00  0.98  0.00  0.00  0.00  0.00  0.98  0.00  0.00
         ITSO237   1.18  0.00  1.18  0.00  0.00  0.00  0.00  1.08  0.00  0.00
         HUB6      6.53  6.53  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
         TCPIP     3.02  0.00  3.02  0.00  0.00  0.00  0.00  3.18  0.00  0.00
14:15:00 TUX8MSTR  2.60  0.00  1.05  0.00  0.00  0.00  0.00  6.83  0.00  0.00
         ITSO232   0.25  0.00  0.25  0.00  0.00  0.00  0.00  0.22  0.00  0.00
         ITSO237   0.82  0.00  0.82  0.00  0.00  0.00  0.00  0.88  0.00  0.00
         HUB6      3.48  3.48  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
         TCPIP     3.65  0.00  3.65  0.00  0.00  0.00  0.00  3.90  0.00  0.00
```

8.11.1 Network errors

ESALPS also utilizes the TUNETCP macro, which provides network error information. This macro checks for up to 50 different errors on each node being measured. When network slowdown is perceived, executing this macro from either a CMS ID or from a Web browser will show all errors detected using the SNMP data source.

8.12 Server resources

Each server has resource requirements that are needed for both understanding server performance, and for projecting the capacity requirements and growth of the server. The performance characteristics of each server that you will want to monitor are: storage, processor, DASD I/O, network traffic, swap, and probably a few more.

The next step beyond measuring server requirements is to monitor individual applications. When the resource requirements of each application are known, then the growth of each application allows for more accurate capacity planning.

Capacity planning allows you to plan price performance, and to provide Service Level Agreements. In the following sections, we provide suggestions on what application and server data you should monitor and why, in order to ensure optimal performance.

8.12.1 Resources by application

Building a profile of an application allows for accurate capacity planning. Knowing the characteristics of an application also allows you to know when current operational characteristics are out of the normal range, suggesting a problem. Storage and processor requirements by application should be well known.

8.12.2 Resources by server

Each server's resource (Storage, Processor and I/O) requirements should be measured. Detecting variations in server requirements can be done with very little overhead. Determining that a server is outside the normal range of operation early means that problem resolution will take less time.

Example 8-11 shows the processor time and storage profile of the top few users on this test/development system. By reviewing this data from your production workload, you'll have an idea of what servers will top the list (using more CPU than other servers), which servers typically use a lot of storage, and how much. There are additional displays showing I/O data by user ID, as well.

Example 8-11 Processor time and storage profile - top users

```
Screen: ESAUSR2  ITSO                          ESAMON V3.1  08/08 17:29-17:30
1 of 3  User Resource Utilization              USER *               2064 40ECB

                   <------CPU time-------> <----Main Storage (pages)----->
         UserID    <----(seconds)----> T:V <Resident> Lock <---WSSize---->
Time     /Class     Total      Virt     Rat Total Actv  -ed Total Actv Avg
-------- -------- ---------- -------- --- ----- ---- ---- ----- ---- ----
```

```
17:30:00 System:      36.088   35.354 1.0  371K 371K  465  412K 412K  11K
         TUX60000     31.641   31.528 1.0 23302  23K    0 22303  22K  22K
         TUX60001      2.069    2.021 1.0 22159  22K    0 24576  24K  24K
         VMLINUX7      0.283    0.255 1.1 31360  31K   30 32768  32K  32K
         VMLINUX1      0.223    0.190 1.2 19198  19K    0 24110  24K  24K
         ESAWRITE      0.195    0.193 1.0  1357 1357    1  1356 1356 1356
         TUX60002      0.190    0.150 1.3  7483 7483    0  7218 7218 7218
         VMLINUX4      0.186    0.156 1.2 23275  23K   31 23244  23K  23K
         VMLINUX9      0.171    0.141 1.2  6195 6195    0  6076 6076 6076
```

8.12.3 Resources by accounting

Service Level Agreements include caps on resources used by the customer. If a customer has multiple servers, you will want an easy way to monitor those resources by customer. The monitor can be used as the capture ratio (the amount of resource accounted for divided by the amount of resource used) is normally above 99% when using ESAMON to capture the data. The monitor data contains accounting codes from the CP directory. All of the performance data can then be reported by accounting code.

8.13 Alerts

Automating detection of problems is important when you have hundreds or thousands of servers. Your system will run much better if problems are detected early. ESAMON provides a large set of alerts defined in a file called EXCPN ALERTDEF. The alert function can be started by any user with access to ESAMON with the command **ESAMON ALERT**. Examples of alerts included by default are:

- Large virtual machine storage sizes and working sets.
- High virtual machine processor utilization, spool consumption, page rate, and I/O rate.
- Looping user detection.
- Idle user detection.
- Missing user detection to ensure all required users are online. The required users are defined in the file named MISSING USER.
- Missing DASD detection to ensure all required DASD are online. The file MISSING DASD provides the list of required volume serials.
- High system page rates.
- Storage offline.
- IDASD utilization and rates.
- Processor utilization.

Alerts can be defined by the installation. Following are some that you may want to add:

- Excessive resources by accounting number
- Eligible lists
- IP Traffic above a specific limit
- Linux swap rates and virtual disk pages resident
- Buffer cache excessive – this leads to high storage requirements
- MDC "not allowed" – fair share for MDC exceeded

8.13.1 Defining and modifying alerts

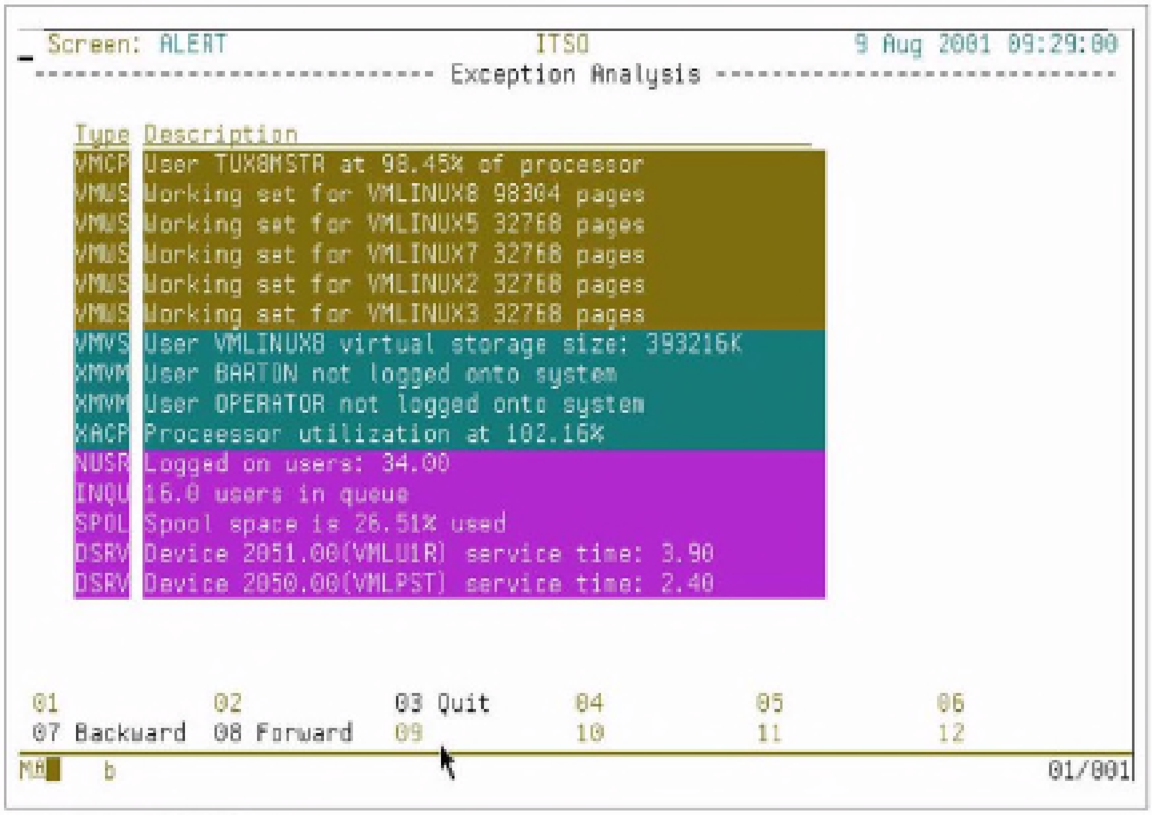
Each ESAMON alert is defined in a file called EXCPN ALERTDEF, which contains multiple alerts such as the following defined. Example 8-12 defines an alert with code VMCP, which will show users above 5% in blue, users above 10% in blue reverse video, and users above 15% in yellow (the latter also has a warning message sent to the operator).

The text of the alert message is defined with the text operand. The EXCPN ALERTDEF file can (and should) be tailored to meet installation needs. It was designed for a CMS interactive environment supporting thousands of users.

Example 8-12 An alert definition

```
ALERT CPUUTIL VMCP
LEVEL1 5 BLUE
LEVEL2 10 BLUE REV
LEVEL3 15 YELLOW REV ACTION CP MSG OP &USERID RUNNING CRAZY
text User &userid at &cpuutil% of processor
```

Figure 8-3 on page 177 shows a user using excessive CPU; 5 users with excessive working sets; a user that was idle for an extended period; users that should be logged, but are missing; spool utilization greater than 20%; and more. Thresholds were set low for this example; in your case, you'll want to edit the exception definition file to meet your installation's requirements. Each type of alert can be a different color, reverse video, and/or blinking, as a way to emphasize specific problems.



```
_ Screen: ALERT                    ITSO                  9 Aug 2001 09:29:00
----------------------------- Exception Analysis -----------------------------

    Type Description
    VMCP User TUX8MSTR at 98.45% of processor
    VMWS Working set for VMLINUX8 98304 pages
    VMWS Working set for VMLINUX5 32768 pages
    VMWS Working set for VMLINUX7 32768 pages
    VMWS Working set for VMLINUX2 32768 pages
    VMWS Working set for VMLINUX3 32768 pages
    VMVS User VMLINUX8 virtual storage size: 393216K
    XMVM User BARTON not logged onto system
    XMVM User OPERATOR not logged onto system
    XACP Processor utilization at 102.16%
    NUSR Logged on users: 34.00
    INQU 16.0 users in queue
    SPOL Spool space is 26.51% used
    DSRV Device 2051.00(VMLU1R) service time: 3.90
    DSRV Device 2050.00(VMLPST) service time: 2.40


 01            02            03 Quit       04            05            06
 07 Backward   08 Forward    09            10            11            12
MA    b                                                               01/001
```

Figure 8-3 ESAMON alert

8.14 Service Level Agreements

You will need to include documentation on service measurement facilities if you provide Service Level Agreements to users or customers. The requirements should include what to report, and how to report it. With today's technology, many customers will want to use a Web-based application to view their data. You should evaluate your performance reporting needs and be prepared for service level reporting.

From a measurement perspective, you will want to monitor resources by accounting number, assuming the Service Level Agreement can be matched to one. Availability of each server should also be monitored.

ESATCP provides alerts to a designated user, which could be to the operator or to a special service machine that has been set up with a PROP function. Each Linux node that will be monitored can have a designated virtual machine user ID that will be alerted for error messages and for a lack of responsiveness.

The terms to include in your Service Level Agreements should define the service being provided, the costs of those service, and escalation procedures, as follows:

- Resource consumption by consumer, minimum guarantee, and cap
- Resource reporting mechanism, report details, reporting granularity
- Alert definition, who is to be alerted, and responses to be taken
- Availability guarantee, reporting mechanism, and reporting granularity

8.14.1 Availability alerts

There are multiple methods you can use to measure availability. Unfortunately, the most common is to measure the server as being "available" unless a user has called in to complain! When running a service environment, knowing when servers are down is more important. The technologies used can be something like ping, which checks a server for response on a regular basis.

ESATCP provides an alert function using SNMP; each node that you have defined to ESATCP can be monitored for availability at a granularity of your choice. Setting the AVAILTIME parameter to as low as 5 seconds will cause a message to be sent every 5 seconds. When no response for 5 seconds is perceived, an alert will be sent to the designated user. (Note that this measures SNMP responsiveness, and not the applications.)

This is suitable for high level availability where potential for losing connectivity and/or the server itself exists. One application may still have failed without impacting other applications or SNMP. As you develop more requirements for availability, you should have tools that test each application. For an example, refer to the discussion on NetSaint in 13.7.1, "NetSaint" on page 323.

8.14.2 Cost of measuring availability

Whichever method of measuring availability you choose, ensure that the cost is minimal. Stories of disabling monitors and having network traffic drop by a significant percent are not uncommon. The cost of using SNMP for testing availability to a server at each designated interval is two UDP packets on the network, each less than 100 bytes.

8.14.3 Availability reporting

The PDB provided by ESALPS records the amount of time a server is up during each interval. The default interval is 60 seconds, with hourly summaries. The following PDB extract provides an hourly summary of availability.

```
EXTRACT:
RECTYPE='SU'
X = 'STOPTIME'
Y = 'HSTSYS.UPTIME'
```

8.14.4 Measuring service

Each server or customer will have agreed-upon resource access—meaning that the server is guaranteed some amount of processing power and some number of I/O each interval. The interval could be per hour, or even per minute.

The service consumed is provided in the ESALPS Performance Data Base. In addition, the wait states of these servers are provided as well. The monitor samples each server to determine what the server is waiting for (this could be page wait, CPU wait, idle or several other states).

For this kind of analysis, the monitor should use a sampling rate of .1, or 10 times per second. The default setting used by ESALPS is normally a rate of 1 per second. This should be changed during the ESALPS installation.

8.15 Measurement function installation

For measuring Linux using NETSNMP, you will need to install NETSNMP on each server you wish to measure. ESALPS is installed on z/VM.

8.15.1 ESALPS installation

ESALPS is made up of ESATCP, ESAMAP, ESAMON, and ESAWEB. They include the support for NETSNMP data, as well as standard MIB-II data. These products are installed per directions that are provided with the products. Personnel are available for on-site installation assistance as well.

Each node (Linux or otherwise) that you wish to monitor will need to be defined to ESATCP as a node file. This file includes the IP address and the community name.

8.15.2 NETSNMP installation

Installing NETSNMP is documented in 13.6.2, "SNMP installation" on page 306. Configure ESATCP with the password (community name) you have coded, and then restart ESATCP. If SNMP is installed with the virtual machine name in the SYSTEM description, you'll be able to easily match IP Node with the virtual machine when the virtual machine is operating under VM.

8.16 Measurement methodology

With measurement data provided from any Linux platform, and in fact any UNIX that runs NETSNMP, you can compare application requirements from one platform to another. This will assist you in choosing which applications to run on zSeries and S/390.

8.16.1 Measuring Linux applications

Each application running on a server (virtual or dedicated) will have different resource requirements. Using the host software resource report (ESAHST1) provided by ESALPS, you can determine the resource requirements of each application.

In Example 8-13, sampling everything from SNMP over a period of 3 minutes, the SNMP Daemon used 0.93% of this Linux (on S/390) server. The HTTP servers are using 8.8 MB each, and SNMPD is using 2468 K.

Example 8-13 LINUX HOST Software Analysis Report

```
Screen: ESAHST1  ITSO                            ESAMON V3.1  08/07 16:46-16:47
1 of 1  LINUX HOST Software Analysis Report     NODE * LIMIT 500     2064 40ECB

                  <--Software Program---------> <CPU Seconds> CPU   Storage(K)
Time     Node     Name      ID    Type   Status  Total Intrval Pct   Current
-------- -------- -------- ----- ------ -------- ----- ------- ----- ----------
16:47:00 ITSO237  httpd        0      0        0     4    0.00  0.00       8836
                  snmpd        0      0        0   229    1.67  0.93       2468
                  sulogin      0      0        0     0    0.00  0.00        448
                  httpd        0      0        0     4    0.00  0.00       8836
                  httpd        0      0        0     5    0.00  0.00       8692
                  inetd        0      0        0     0    0.00  0.00        576
```

8.16.2 Measuring Linux server requirements

Each server may run many applications. The sum of the resource requirements of these servers impact the total storage requirements, the amount of swap space required, and the processing power requirements. The ESAHST1 report provides, by application, the processor and storage requirements of each application. These values should be followed over peak periods to show how they would impact other workloads if moved to a zSeries or S/390. Using the abbreviated sample from Example 8-13, this server uses about 30 MB for the identified applications.

In Example 8-14 and Example 8-15, an ESAUCD2 real-time example, you can then measure the total amount of storage that Linux has allocated. The first screen shows the amount of real storage and swap storage available and in use by Linux. When using Linux under z/VM, you'll want to minimize storage requirements.

Alerts can and should be set to show when Linux guests use swap, and when the buffer exceeds some threshold Using swap indicates the need for more memory, while a large buffer indicates too much storage.

Of the 512 MB defined on this system (516328 K), about 480 MB is accounted for between the shared storage, the buffer storage, the cache storage and the "used" storage. Reducing the size of the ITSO232 machine by 400 K would have no impact on the applications currently in use.

Example 8-14 LINUX UCD Memory Analysis Report (screen 1 of 2)

```
Screen: ESAUCD2  ITSO                             ESAMON V3.1  08/08 14:35-14:55
1 of 2  LINUX UCD Memory Analysis Report          NODE * LIMIT 500     2064 40ECB

                  <--Real Storage--> <-----SWAP Storage----> Total
Time     Node     Total  Avail Used  Total Avail Used  MIN   Avail
-------- -------- ------ ----- ----- ----- ----- ----- ----- -----
14:55:00 ITSO232  516328  457K 58960  143K  142K  1360 16000 58960
         ITSO237  257000  176K 80524  143K  143K     0 16000  320K
14:54:00 ITSO232  516328  457K 58960  143K  142K  1360 16000 58960
         ITSO237  257000  176K 80524  143K  143K     0 16000  320K
14:53:00 ITSO232  516328  457K 58968  143K  142K  1360 16000 58968
         ITSO237  257000  176K 80524  143K  143K     0 16000  320K
14:52:00 ITSO232  516328  457K 59100  143K  142K  1360 16000 59100
         ITSO237  257000  176K 80524  143K  143K     0 16000  320K
```

Example 8-15 LINUX UCD Memory Analysis Report (screen 2 of 2)

```
Screen: ESAUCD2  ITSO
2 of 2  LINUX UCD Memory Analysis Report

                  <--Storage in Use-> Error
Time     Node     Shared Buffer Cache Message
-------- -------- ------ ------ ----- -----------
14:52:00 ITSO232   37788 378112  9432
         ITSO237   28412  28952 10216
14:51:00 ITSO232   38380 378112  9432
         ITSO237   28412  28952 10216
14:50:00 ITSO232   38380 378112  9432
         ITSO237   28412  28952 10216
```

8.16.3 Measuring VM Virtual Machine

On VM, the resource reports (ESAUSR2, ESAUSR3, ESAUSR4) show the resources of the virtual machine. When analyzed, the data from the ESAHST1 display closely matched the processing requirements reported against the virtual machine by VM on the ESAUSR2 display.

8.17 Tuning guidelines

In the following sections, we provide miscellaneous configuration guidelines that will help you avoid problems and bypass errors that may be not be obvious to installations installing VM for the first time.

8.17.1 Paging and spooling (one extent per Real Device Block)

There should be only *one* page or spool extent per volume. Having multiple extents adds to overhead and may degrade performance. Both spool and page I/O have been optimized with a "never-ending channel program" that allows I/O to bypass the overhead of starting I/O. By having volumes with different types of data, there is an added overhead for each I/O of stopping one I/O and then starting another.

8.17.2 Enterprise Storage Server (ESS)

For VM, you'll want to define as many Real Device Blocks as possible; only use 3390-9 emulation when absolutely necessary.

The issue with current DASD caching technology is based on a large percent of the I/O being handled by the cache. Under z/VM (at least through V4.1), there is only one real device block per logical disk—and there is a restriction that only one I/O can be started at a time to each logical disk.

Thus, if you have a very large amount of data on a single volume, then when one I/O must retrieve data from the disk, no other I/O can be started even for data that is currently residing in the cache. OS/390 supports large volumes using Parallel Access Volumes (PAV). This allows OS/390 to have multiple I/O to a single logical device by defining multiple paths to a device.

Thus, with PAV, a logical device can be busy, and the data that's being cached for that device by the ESS is available on other paths. Z/VM does not support this. Without PAV support, you will get optimum performance with smaller and more device addresses.

However, there are two considerations:

1. Large files that are accessed by a single task are not impacted by not having duplicate paths to data. The task must wait for an I/O to complete before starting another one.
2. Large numbers of small servers with random I/O should have the ability to have concurrent I/O. When configuring your storage controller, maximize the number of concurrent I/O by maximizing the number of logical devices.

8.17.3 Virtual machine sizes

Reduce virtual machine sizes as much as possible. Tailoring a server to a specific application and minimizing its storage requirements will mean more storage is available for other servers.

8.17.4 DASD format

The DASD formats and I/O drivers are documented in 8.10, "DASD subsystem" on page 161. You should measure I/O response times, the impact of using MDC, and DASD cache to determine if you are getting the most out of your DASD subsystem.

With `dasdfmt`, MDC only works with track cache, and you must use the ECKD driver that does start subchannels.

The CMS Format Reserved has several advantages, both operationally and from a performance perspective. Either the DIAG driver (Diagnose250) or the ECKD driver may be utilized. When using the DIAG driver, MDC can utilize record level caching. For applications with 4 K blocks read in a random fashion, record level caching will read in just 4 K records instead of full tracks of data. This can reduce the I/O time, channel delays, and storage requirements. Operationally, the files may be read using CMS utilities such as backup.

8.17.5 MDC: fair share considerations (NOMDCFS)

Servers that provide data to other servers should have OPTION NOMDCFS in their directory. MDC is managed with a fair share algorithm that will disallow data to be inserted in to MDC by users that have exceeded their share. For some servers, however, using fair share is inappropriate—so for these servers, put OPTION NOMDCFS in their CP directory entry.

8.17.6 Swap: RAMdisk vs. virtual disk

To reduce Linux storage (by reducing the amount of storage Linux will use for caching data, yet still have enough storage to meet operational requirements), two methods are available:

1. You can define a RAMdisk as part of Linux virtual storage and use this as swap. Linux will then only use this storage for a swap disk.
2. You can use the VM virtual disk facility. By defining a virtual disk and then telling Linux to use it as a swap device, you have a swap device in storage.

When using virtual disks for swap, the important measurement is the amount of storage being used by the swap disk, from the VM perspective. When the amount of storage used for swap becomes large as compared to the virtual machine size, you are likely incurring overhead of moving pages from swap to Linux main storage (which is a cost in processing time).

While no rules of thumb have been developed for this yet, you could assume that controlling the rate of Linux swap activity should be the secondary objective, with the primary objective being to reduce the total storage (virtual machine plus virtual disk) requirements. The ESAVDSK real time display provided by ESAMON shows exactly how much storage is in use for the virtual disk. The ESAVDSK report produced by ESAMAP shows the same information, but normally over a longer period of time.

The following ESAVDSK is an example of a virtual disk used for swap for the duration of a single task. When there was a requirement for more storage, the virtual disk was used. After it was no longer needed, the virtual disk was paged out.

```
Screen: ESAVDSK  Velocity Software, Inc.        ESAMON V3.1
                                           <--pages--> DASD   X-
                                           Resi- Lock-  Page Store
Time     Owner    Space Name               dent    ed Slots  Blks
-------- -------- ------------------------ ----- ----- ----- -----
12:15:01 LINUX001 VDISK$LINUX001$0202$0009    36     0    50     0
12:16:01 LINUX001 VDISK$LINUX001$0202$0009    36     0    50     0
12:17:01 LINUX001 VDISK$LINUX001$0202$0009   173     0    50     0
12:18:01 LINUX001 VDISK$LINUX001$0202$0009   293     0    35     0
12:19:01 LINUX001 VDISK$LINUX001$0202$0009   293     0    35     0
12:39:01 LINUX001 VDISK$LINUX001$0202$0009   259     0    35     0
12:40:01 LINUX001 VDISK$LINUX001$0202$0009   259     0    35     0
12:41:01 LINUX001 VDISK$LINUX001$0202$0009   207     0    86     0
12:42:01 LINUX001 VDISK$LINUX001$0202$0009   207     0    86     0
12:43:01 LINUX001 VDISK$LINUX001$0202$0009    13     0   280     0
12:44:01 LINUX001 VDISK$LINUX001$0202$0009    13     0   280     0
12:45:01 LINUX001 VDISK$LINUX001$0202$0009    13     0   280     0
```

8.17.7 Timer tick kernel changes

The currently available Linux for zSeries and S/390 distributions implement the timer functions in the kernel using a 10 mS interrupt that increments the "jiffies" global variable. Work is being done on an implementation that is more suitable to Linux running as guests under VM.

When planning on operating many Linux servers under z/VM, you should plan on implementing this "no more jiffies" patch as soon as it is available. This reduces the processor requirements of supporting many idle guests. Without this patch, you can expect 0.2 to 0.3% of a processor (G5) to be used by each idle server.

8.17.8 Kernel storage sharing

Reducing storage requirements by sharing storage is used by CMS for the CMS operating system, by programs, and for data. This technology is slowly being developed for Linux. We suggest you watch for developments and implement them when possible.

Part 2

Practical considerations

In this part of the book we provide explicit examples of the work we did during this residency. For a theoretical discussion of the concepts behind installing and managing z/VM and Linux for zSeries and S/390 systems, see Part 1, “Theoretical considerations” on page 1.

9

VM configuration

Running a large number of Linux virtual machines on a VM system is a serious challenge, with many configuration and tuning issues to deal with in order to make the system work properly. Some of these issues are related to VM, and some are Linux issues. In this chapter, we focus on the VM aspects of the configuration.

9.1 General VM configuration issues

Using an average VM system straight out of the box, you will probably not be able to run a large number of Linux images efficiently, so you should be aware of the following general guidelines.

9.1.1 Allocate sufficient paging space

Virtual machines running Linux tend to be rather large (compared to average CMS users), so be prepared to have sufficient paging DASD set up to accommodate all the virtual storage that you give out to your users. Unlike with CMS users, over time virtual machines running Linux will use all the storage you give them.

The recommendation currently is to have twice as much paging space on DASD than the sum of your total virtual storage, in order to let CP do block paging. This means that to run 10 Linux virtual machines of 512 MB each in an LPAR with 2 GB of main storage, you need to have 14 GB worth of paging DASD. Failure to do so may cause your VM system to take a PGT004 abend before you notice paging space filling up.

With this amount of space you will not be tempted to mix it with other data, but it should be clear you do not mix paging space with other data on the same volumes. If you plan to use saved systems to IPL Linux images as outlined in this chapter, you also need to seriously evaluate spooling capacity, because NSS files reside on the spool volumes. See 8.17.1, "Paging and spooling (one extent per Real Device Block)" on page 182 to learn why this is important.

9.2 Things to do for new Linux images

When creating new Linux images, the following tasks need to be performed.

9.2.1 Create a central registry

In order to manage a large number of Linux images on a VM system, you need to maintain a central registry of the Linux images and use standard processes to create the user IDs. Several of the utility services on VM would need to do the correct things to these images, based on the registration.

For the purpose of this discussion, it does not really matter whether this "central registry" is your white board in the office or a few files on a shared disk somewhere in the system. (However, ease of access and the options for automation make it attractive to use online files to hold the configuration data.)

9.2.2 Create the user ID in the CP directory

The CP directory entry only defines the virtual machine, but many of the parameters (such as storage) that define the virtual machine can be specified or overruled at startup time. Also, many aspects of the virtual machine, such as performance settings, cannot yet be specified in the CP directory and therefore need to be handled by automation software. Other aspects, such as IUCV authorization, do need to be defined in the CP directory. The point is that if, in your installation, many things need to be arranged outside the CP directory, it is questionable whether Linux images need to be bound to specific VM user IDs.

Compare this to an installation with discrete servers where these servers do not have a hard disk and boot from the LAN. The tables in bootp will define what image needs to run on what hardware. This makes it easy to deal with hardware failures and so on.

You could adopt a similar strategy in VM and use the parameters that *must* be in the CP directory to help you determine what user ID to use.

9.2.3 Allocate the minidisks

When you create a Linux image, you need to allocate and initialize the disks for the Linux image. You need to plan this carefully so that you can place the minidisks on the proper volumes.

9.2.4 Define the IP configuration

Each Linux image will need TCP/IP connectivity. Apart from getting an IP address for the system and having it registered in the Domain Name System (DNS), the Linux image will also need a network connection. Depending on the network architecture used, this means you must define the IP address in the OSA configuration or prepare the point-to-point connection in the VM TCP/IP or Linux hub.

9.2.5 Install and configure the Linux system

When the user ID is defined in VM and TCP/IP characteristics are known, the Linux system can be installed and configured. Note that, with a number of similar Linux images running on the same VM system, there are more efficient ways to get another working Linux image than just run the entire installation process; these issues are addressed in Chapter 10, "Cloning Linux images" on page 209.

9.2.6 Register the user ID with automation processes

This registration will ensure that the user ID is automatically started, stopped, monitored, and so on.

9.2.7 Register the user ID so backups can be made

This registration should also ensure recovery will be done when something "breaks".

9.3 Using VM TCP/IP as the virtual router

The VM TCP/IP stack can be used as a virtual router. With the current restrictions of the CTC and IUCV drivers in Linux (see 17.1, "Ability to reconfigure CTC and IUCV" on page 408), there are advantages in using the VM TCP/IP stack.

For example, new interfaces and point-to-point links can be defined and activated on the VM TCP/IP stack without stopping and starting it. However, the process to do this—as well as the syntax of the configuration files—can be slightly intimidating. The following section demonstrates how to use the OBEYFILE command to dynamically create the connections. The topology of the network is shown in Figure 9-1 as a guide to the IP addresses given in the examples.

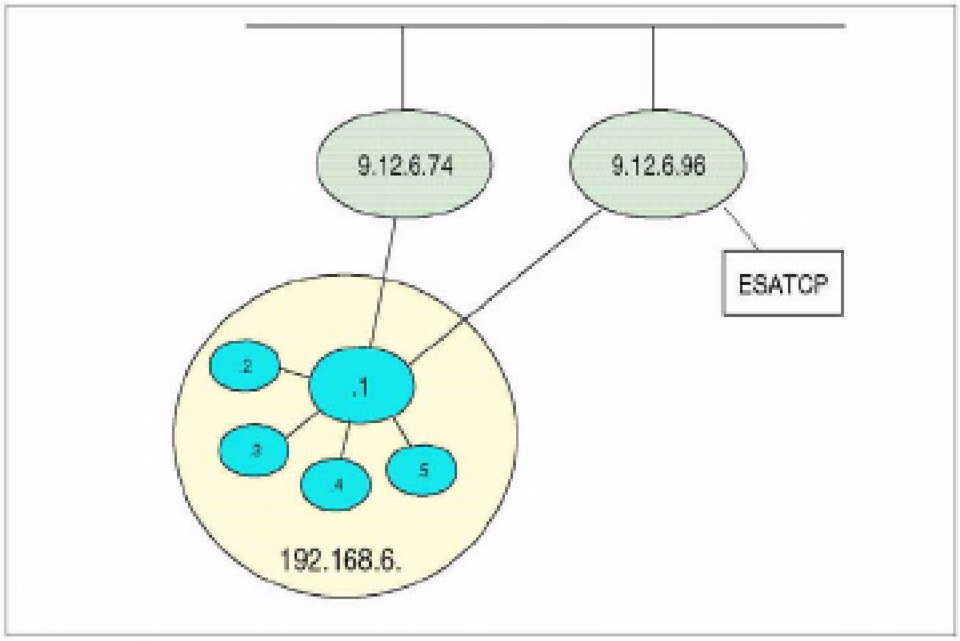


Figure 9-1 Topology of the network

The scenario described here is for IUCV connections. The same process can be done for CTC connections, with a few minor differences. The first difference is in the DEVICE and LINK statements (see *TCP/IP Planning and Customization*, SC24-5981, for the details). The other difference is that virtual CTC devices need to be defined in the VM TCP/IP stack virtual machine and COUPLEd to the corresponding other virtual CTC device.

9.3.1 Dynamic definitions and the PROFILE TCPIP file

When the VM TCP/IP stack is started, it reads the configuration from its profile. When you change the configuration of a running VM TCP/IP stack with the OBEYFILE command, you must also update the profile to make sure these changes will be picked up when the VM TCP/IP stack is restarted.

The OBEYFILE command is unlike other VM commands. When a user issues the OBEYFILE command, that causes the VM TCP/IP stack link to the minidisk of that user. It then reads the file specified on the OBEYFILE command from that disk to pick up the configuration statements. This means the VM TCP/IP stack must be authorized to link to the user's disk (either by the ESM or through a valid read-password).

Note: If no ESM is used on VM, the read-password must be supplied as an option for the OBEYFILE command. It is annoying that the OBEYFILE command parses the parameters and options different from normal CMS commands. The read-password (specified as an option after the "(" character) is only recognized when filetype and filemode of the file are specified.

A "naive" implementation of OBEYFILE can cause problems with ad hoc changes to the TCP/IP configuration. If the program is invoked by an automated process, most of these problems can be avoided.

Note: Now that the QUERY MDISK command with the USER option is available even for general users, we believe it should be considered a bug that the VM TCP/IP stack does not use this for the OBEYFILE command. With the current implementation, it is quite possible for a VM TCP/IP stack to link an incorrect disk and activate the wrong configuration statements.

Because you also need to update the TCP/IP profile, you want that disk to be linked R/O by the VM TCP/IP stack. This way the user that issues the OBEYFILE commands can also update the profile.

9.3.2 Creating the device and link

The point-to-point connection in VM TCP/IP requires both a device and a link to be defined in the PROFILE TCPIP file:

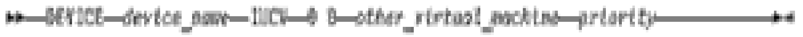


Figure 9-2 The DEVICE statement for an IUCV connection

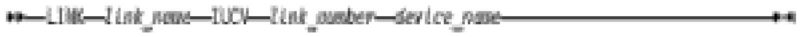


Figure 9-3 The LINK statement for an IUCV connection

While having device names and link names promotes flexibility, for point-to-point connections it can become quite cumbersome.

Fortunately TCP/IP doesn't care if we use the same identifier both for the device name and for the link name[1]. And since the VM TCP/IP stack has just a single point-to-point connection to each Linux image, we might as well use the user ID of the Linux virtual machine as device name and link name. Since the parsing rules for the TCP/IP profile do not require the DEVICE and LINK statement to be on different lines, we can put both on the same line. The syntax may look a bit redundant, but this makes it much easier to automate things.

The definition in the TCP/IP profile for a point-to-point link to our point-to-point connections can now be defined as shown in Example 9-1.

Example 9-1 The DEVICE and LINK statements for our virtual router

```
device tcpip    iucv 0 0 tcpip    a link tcpip    iucv 0 tcpip
device vmlinux6 iucv 0 0 vmlinux6 a link vmlinux6 iucv 0 vmlinux6
device tux80000 iucv 0 0 tux80000 a link tux80000 iucv 0 tux80000
device tux80001 iucv 0 0 tux80001 a link tux80001 iucv 0 tux80001
device tux80002 iucv 0 0 tux80002 a link tux80002 iucv 0 tux80002
device tux80003 iucv 0 0 tux80003 a link tux80003 iucv 0 tux80003
```

[1] There is no concern that future versions of VM TCP/IP will be more strict in this aspect.

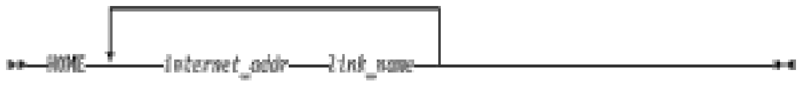
The first connection is the "uplink" to the VM TCP/IP stack which we used to give ESATCP (part of the Linux Performance Suite) access to the Linux images. The second one provides a direct connection to another Linux image on the same VM system to allow clients in that Linux image (e.g. telnet) to access the Linux images. This connection would not be possible via the real network. See 4.4.3, "Using the OSA with Linux" on page 95 for more details.

If you compare this with the syntax definition for the DEVICE and LINK statement, you can deduce which occurrence of the user ID is what. (This may seem confusing, but will be worth the effort because you won't need to remember whether the START statement requires the link name or the device name.)

9.3.3 Defining the home address for the interface

The IP address of the VM TCP/IP stack side of the point-to-point connection must be defined in the HOME statement.

Figure 9-4 The syntax of the HOME statement



For point-to-point connections, the same IP address can be specified for each link. Since the IP address does not have to be in the same subnet as the other side of the connection (we specify a subnet mask of 255.255.255.255), we can even use the uplink IP address for it.

The virtual router in our example does not have its own real network interface either, but uses an IUCV connection to the main VM TCP/IP stack. The first part of the HOME statement in our profile is shown in Example 9-2.

Example 9-2 The HOME statement in the TCP/IP profile

```
home
  192.168.6.1   tcpip
  192.168.6.1   vmlinux6
  192.168.6.1   tux80000
  192.168.6.1   tux80001
  192.168.6.1   tux80002
  192.168.6.1   tux80003
```

Unfortunately, VM TCP/IP requires all interfaces to be listed in a single OBEYFILE operation when a new interface is added.

9.3.4 Defining the routing information

The GATEWAY statement is used to specify the IP address of the stack at the other side of the point-to-point connection.

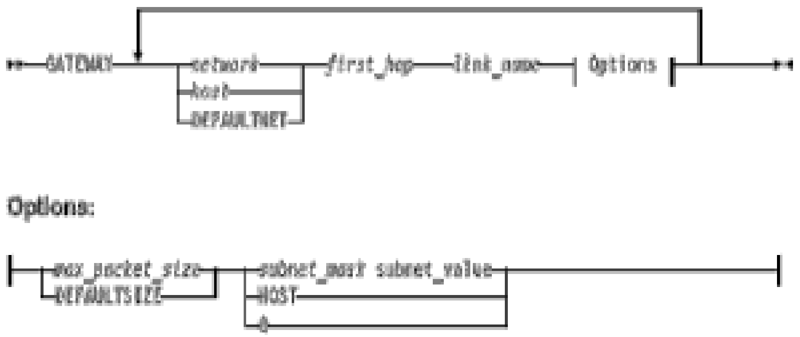


Figure 9-5 The syntax of the GATEWAY statement

As shown in Figure 9-5, each of the connections must be listed in the GATEWAY statement. Example 9-3 shows the routing statements for our virtual router. The tcpip link is the uplink connection to the VM TCP/IP stack.

Example 9-3 The GATEWAY statement in the profile

```
GATEWAY
; (IP) Network  First        Link      Max. Packet  Subnet      Subnet
; Address       Hop          Name      Size (MTU)   Mask        Value
; -----------   ------------ -------   -----------  ----------- --------
  9.12.6.96     =            tcpip     8188         host
  9.12.6.74     =            vmlinux6  8188         host
  defaultnet    9.12.6.74    vmlinux6  8188         0

  192.168.6.2   =            tux80000  8188         host
  192.168.6.3   =            tux80001  8188         host
  192.168.6.4   =            tux80002  8188         host
  192.168.6.5   =            tux80003  8188         host
```

The defaultnet route is to the Linux machine vmlinux6 (instead of to the VM TCP/IP stack, as you might expect); this is because the 192.168 addresses only exist on this VM system. Trying to let one of these images connect to another host in the 9. network wouldn't work because there is no route back into this VM system.

Just as with the HOME statement, VM TCP/IP requires the entire GATEWAY statement to be supplied in the OBEYFILE command when something must be changed or added to it.

9.3.5 Starting the connection

To start the point-to-point connection, the START command is given for the device.

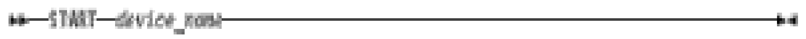


Figure 9-6 The syntax of the START command

A single START command is used to start the connection for the device. The TCP/IP profile must have START commands for all devices that you want to start. If you are defining the devices and links up front, it may be an advantage to postpone the START until the Linux guest is ready to connect. Otherwise you would waste time (and console log lines) with the VM TCP/IP stack retrying that connection.

Retry and reconnect

The connection between two TCP/IP stacks via IUCV consists of two IUCV paths. Each side will "connect" its sending connection to the other party. For a working IP connection, both paths need to be up. When the sending IUCV path is "severed" by the other end, the stack will respond by "severing" the other path as well.

There is difference between the way the VM TCP/IP stack and the Linux netiucv driver handle the connection setup. When an existing connection is stopped from the VM side, Linux will notice that and bring the link down as well (and show that in the system log.

However, when the link is started again from the VM side, Linux will not pick up the connection request. Even an **`ifconfig iucv0 up`** command will not do the trick because the network layer assumes the connection is still up. To make Linux take action, you need to execute **`ifconfig iucv0 down`** followed by an **`ifconfig iucv0 up`** command. Unfortunately, that removes the default route you may have set up using that connection. We believe the Linux netiucv driver should be changed to listen for a connection attempt again after the path was severed.

If the VM TCP/IP stack is retrying the link, it will attempt to connect to the other side every 30 seconds. Between those attempts, the VM TCP/IP stack is "listening" all the time and will respond immediately when Linux tries to establish a connection.

For a CTC connection, there also is a retry every 30 seconds when the connection is down. The process is slightly different in that the VM TCP/IP stack does not have a read outstanding all the time when the connection is waiting to be retried. This means that when a new Linux image is brought up, it may need to wait up to 30 seconds before the VM TCP/IP stack is able to take notice of the attempt. This waiting period may seem trivial, but it really is not when you talk about bringing up a new image in 90 seconds.

Attention: We believe there are problems with the way the Linux CTC driver handles the reconnect. More than once we found Linux in a tight loop, trying to restore a broken CTC connection. We did not have the time to dig into these problems. Casual debugging of this with the CP TRACE command is difficult with the 10 mS timer tick going on.

The 2.4.5 version of the kernel appears to be changed in using a shorter period to wait for a response from the other side. That period could be too short, since we were frequently unable to restart a failing connection.

9.3.6 Putting all the pieces together

A simple program was written to add the connection for a Linux image to the VM TCP/IP stack. It updates the PROFILE TCPIP and issues the OBEYFILE command to activate the new connection on the running VM TCP/IP stack. To make the file easier to parse, we added a few special comments in the file to mark the place where items should be inserted. Our TCPIP PROFILE is shown in Example 9-4.

A generic parsing routing for the configuration file is complicated, but we do not need this flexibility when the new entries are added through an automated process anyway.

Example 9-4 The PROFILE TCPIP for our virtual router

```
tinydatabufferpoolsize      20
monitorrecords
timestamp prefix

device tcpip    iucv 0 0 tcpip    a link tcpip    iucv 0 tcpip
device vmlinux6 iucv 0 0 vmlinux6 a link vmlinux6 iucv 0 vmlinux6
device tux8mstr iucv 0 0 tux8mstr a link tux8mstr iucv 0 tux8mstr
device tux80000 iucv 0 0 tux80000 a link tux80000 iucv 0 tux80000
device tux80001 iucv 0 0 tux80001 a link tux80001 iucv 0 tux80001
; =device

home
  192.168.6.1   tcpip
  192.168.6.1   vmlinux6
  192.168.6.1   tux8mstr
  192.168.6.1   tux80000
  192.168.6.1   tux80001
; =home

GATEWAY
; (IP) Network  First        Link     Max. Packet  Subnet      Subnet
; Address       Hop          Name     Size (MTU)   Mask        Value
; -----------   ------------ -------  -----------  ----------- --------
  9.12.6.96     =            tcpip    8188         host
  9.12.6.74     =            vmlinux6 8188         host
  defaultnet    9.12.6.74    vmlinux6 8188         0
  192.168.6.254 =            tux8mstr 8188         host


  192.168.6.3   =            tux80000 8188         host
  192.168.6.4   =            tux80001 8188         host
; =gateway

start tcpip
start vmlinux6
start tux8mstr

start tux80000
start tux80001
; =start
```

With the restrictions we've put on the layout of the configuration file, the program to do the updates can be really simple, as shown in Example 9-5. The program adds the proper entries to a copy of the configuration file on the A-disk. It then builds a temporary file with the DEVICE statement, the START statement and the HOME and GATEWAY sections, and issues an OBEYFILE command against that file. When OBEYFILE gives a return code 0, the configuration file is replaced by the new one, and the temporary files are erased.

Example 9-5 Simple program to add a connection to the configuration

```
/* ADDTCPIP EXEC      Add a Linux guest to TCP/IP                   */

parse arg userid nr .

'PIPE state PROFILE TCPIP * | spec w1.3 1 | var config'
workfile = 'TEMP TCPIP A'

'PIPE <' config '|locate /'userid'/ | count lines | var cnt'
if cnt > 0 then
  do
```

```
    say 'Link for' userid 'probably already present'
    return 1
  end

devs = 'device' userid 'iucv 0 0' userid 'a link' userid 'iucv 0' userid
home = '  192.168.6.1 ' userid
gate = '  192.168.6.'left(nr,3) '=          ' userid '8188        host'
strt = 'start' userid

'PIPE (end \)',
  '\ <' config,
  '| x1: strtolabel /; =device/',
  '| i: fanin',
  '| >' workfile,
  '\ var devs | i:',
  '\ x1:',
  '| x2: strtolabel /; =home/    | i:',
  '\ var home | i:',
  '\ x2:',
  '| x3: strtolabel /; =gateway/ | i:',
  '\ var gate | i:',
  '\ x3:',
  '| x4: strtolabel /; =start/   | i:',
  '\ var strt | i:',
  '\ x4: | i:'

'PIPE (end \)',
  '\ <' workfile,
  '| strfrlabel /; =device/',         /* Take HOME and GATEWAY sect  */
  '| strtolabel /; =gateway/',
  '| preface var strt',               /* and the START               */
  '| preface var devs',               /* Add the DEVICE statement    */
  '| >' userid 'TCPIP A'

'OBEYFILE' userid 'TCPIP A (READ'
if rc = 0 then
  do
    'COPYFILE' workfile config '(OLDD REPL'
    'ERASE' workfile
    'ERASE' userid 'TCPIP A'
  end
return rc
```

Deleting a link is also possible. Because of the simple layout of the configuration file, we can do it with a program as shown in Example 9-6 on page 201. Unfortunately, we cannot delete the device statement once it is defined to the VM TCP/IP stack, but we can at least stop it.

Example 9-6 Deleting a connection from the configuration

```
/* DELTCPIP EXEC     Delete a Linux image from TCP/IP configuration */

parse arg userid .
'PIPE var userid | xlate lower | var userid'

'PIPE state PROFILE TCPIP * | spec w1.3 1 | var config'
workfile = 'TEMP TCPIP A'

'PIPE <' config '| nlocate /'userid'/ | >' workfile

'PIPE (end \)',
  '\ <' workfile,
  '| strfrlabel /; =device/',         /* Take HOME and GATEWAY sect  */
  '| strtolabel /; =gateway/',
  '| literal STOP' userid,            /* and the START               */
  '| >' userid 'TCPIP A'

'OBEYFILE' userid 'TCPIP A (READ'
if rc = 0 then
  do
    'COPYFILE' workfile config '(OLDD REPL'
    'ERASE' workfile
    'ERASE' userid 'TCPIP A'
  end
return rc
```

9.3.7 Define and couple the CTC devices

In addition to what is shown in previous sections for point-to-point connections over IUCV, a virtual CTC needs to be defined and coupled. Both DEFINE and COUPLE are CP commands that can be issued through the NETSTAT CP interface. The COUPLE command can be issued from either side of the connection. The **hcp** command (from the cpint package) can be used to issue the COUPLE commands from the Linux side.

To have the virtual CTCs defined at startup of the VM TCP/IP stack, you can define them in the user directory or include them in the SYSTEM DTCPARMS file with the :vctc tag. When the CTC is defined through the DTCPARMS file, the COUPLE command is also issued (provided the Linux machine is already started up). The PROFILE EXEC of your Linux guest should also be prepared to define the virtual CTC (if not done through the user directory) and try to couple to the VM TCP/IP stack, in case Linux is started after the VM TCP/IP stack.

If you run a large number of Linux images this way, you probably should have a control file read by the PROFILE EXEC of your Linux guest to define the proper virtual CTCs. One option would be to read and parse the DTCPARMS control file of TCP/IP so that you have a single point to register the CTCs.

9.4 Using DirMaint to create Linux virtual machines

On a VM system with more than a few user IDs, it is impractical to maintain the user directory manually. Editing the directory by hand is cumbersome and error-prone. Security is another consideration, because if you do not run an External Security Manager for VM, logon passwords and minidisk passwords are also maintained in the user directory—and they are visible in clear text to the person maintaining the directory (and anyone who looks over his or her shoulder).

DirMaint is the IBM program product to manage your VM user directory. Its complete name is "5748-XE4 Directory Maintenance VM/ESA".

If your installation is using DirMaint, you must use DirMaint to maintain the user directory; you do not have the option in that case of managing your Linux user IDs by hand-editing the USER DIRECT (and trying to do so could cause serious problems). The same holds true when the VM installation is using a directory management product from a solution developer like VM:Secure. You cannot realistically run different directory management products on the same VM system.

Using DirMaint is not the only way to manage your user directory; it can also be managed by a CMS application as a simple flat file in CMS and brought online with the DIRECTXA command. If you only have very typical standard Linux images, then maintaining this flat file could be automated fairly easily. However, when you run z/VM to host a large number of Linux images, you are likely to end up with a lot of user IDs that are non-standard or different.

9.4.1 Why to avoid GET and REPLACE

Even when DirMaint is in control of your user directory, you can still mostly maintain it yourself if you want to use GET and REPLACE commands to update user entries in DirMaint. However, we recommend that you learn the DirMaint commands to do incremental directory updates, because this is less error-prone than editing the directory entries by hand. Understanding how to use these DirMaint commands will also give you an idea of how the process can be automated.

An additional bonus for avoiding the use of GET and REPLACE is that the DirMaint console log will give you a full report of each change that was made to the directory.

9.4.2 Keeping the user directory manageable

Both directory profiles and prototype files can be used to simplify management of the user directory.

Directory profiles

The CP user directory supports profiles to be used in the definition for a user in the directory. The INCLUDE directory statement in the user entry identifies the profile to be used for that user. Obviously DirMaint also supports the use of these directory profiles. The profile itself is managed by DirMaint similar to user IDs, so you can use normal DirMaint commands to add statements to profiles.

The profile contains the directory statements that are identical for the user IDs (e.g. link to common disks, IUCV statements). You can create different profiles for the different groups of users that you maintain.

Example 9-7 Sample directory profile

```
PROFILE TUX6PROF
ACCOUNT TUX6
IPL 1B0
IUCV TUX6MSTR
MACHINE XA
CONSOLE 0009 3215 T
SPOOL 000C 2540 READER *
SPOOL 000D 2540 PUNCH A
SPOOL 000E 1403 A
LINK MAINT 0190 0190 RR
LINK MAINT 019D 019D RR
LINK MAINT 019E 019E RR
LINK TUX6MSTR 0191 0191 RR
LINK TUX6MSTR 01A1 01A1 RR
LINK TUX6MSTR 01A0 02A0 RR
LINK VMLINUX6 01B0 01B0 RR
```

The include profile shown in Example 9-7 is used for all Linux images in a penguin colony to allow each of the user IDs to set up an IUCV connection to the “leader of the colony” (which is TUX6MSTR, in this case) and have links to some of the disks of the leader.

Prototype files

A *prototype* is like a skeleton for a user entry in the directory. The prototype is only used by the DIRM ADD command with the LIKE option to create a new user entry according to the prototype.

Example 9-8 Using DIRM ADD with LIKE to add a user ID

```
dirm add tux80004 like tux8 pw icefloes
DVHXMT1191I Your ADD request has been sent for processing.
Ready; T=0.04/0.04 14:58:40
 DVHREQ2288I Your ADD request for TUX80004 at * has been accepted.
 DVHBIU3425I The source for directory entry TUX80004 has been updated.
 DVHBIU3425I The next ONLINE will take place as scheduled.
 DVHSCU3541I Work unit 08145842 has been built and queued for processing.
 DVHSHN3541I Processing work unit 08145842 as RVDHEIJ from VMLINUX,
 DVHSHN3541I notifying RVDHEIJ at VMLINUX, request 75.1 for TUX80004
 DVHSHN3541I sysaffin *; to: AMDISK 01A0 3390 AUTOG 100 LNX MR BLKSIZE
 DVHSHN3541I 4096
 DVHRLA3891I Your DMVCTL request has been relayed for processing.
 DVHDRC3428I Changes made to directory entry DATAMOVE have just been
 DVHDRC3428I placed online.
 DVHDRC3428I Changes made to directory entry TUX80004 have just been
 DVHDRC3428I placed online.
 DVHRLA3891I Your DMVCTL request has been relayed for processing.
 DVHREQ2289I Your ADD request for TUX80004 at * has completed; with RC
 DVHREQ2289I = 0.

 DVHSHN3541I Processing work unit 08145842 as RVDHEIJ from VMLINUX,
 DVHSHN3541I notifying RVDHEIJ at VMLINUX, request 75.1 for TUX80004
 DVHSHN3541I sysaffin *; to: AMDISK 01A0 3390 AUTOG 100 LNX MR BLKSIZE
 DVHSHN3541I 4096
 DVHDRC3428I Changes made to directory entry DATAMOVE have just been
 DVHDRC3428I placed online.
 DVHDRC3428I Changes made to directory entry TUX80004 have just been
 DVHDRC3428I placed online.
 DVHSHN3430I AMDISK operation for TUX80004 address 01A0 has finished
 DVHSHN3430I (WUCF 08145842).
```

The DIRM ADD command in Example 9-8 shows how user ID TUX80004 was added using the prototype TUX8. The USER statement in the prototype file would cause each user ID created from the prototype to have the same password.

To avoid that, DirMaint requires the password for the user ID to be specified on the ADD command. The asynchronous messages from this command also refer to the DATAMOVE user ID. Because the AMDISK statement in the prototype specifies that disk should be formatted, it is handed to the DATAMOVE user ID first to have it formatted. A so-called “workunit” is created to format the disk. The second part of the output shows the messages related to the workunit being created and completed.

Changes that you apply to a prototype file afterward will not affect the user entries created using that prototype. The prototype is used for directory statements that cannot be placed in the profile (like USER and INCLUDE) and MDISK statements that are similar but not identical (although the number of minidisks and their sizes will be the same, the starting cylinder will be different for each Linux image). Automatic allocation (using the AUTOV or AUTOG option) can be used to define the minidisks.

Example 9-9 Prototype file

```
USER TUX8 ICEFLOE  48M 256M G
   INCLUDE TUX8PROF
   MDISK 01A0 3390 AUTOG  100 LNX    MR
```

When a new user ID is added using this prototype file, DirMaint will automatically allocate 100 cylinders in group LNX for the minidisk.

9.4.3 Rotating allocation

Automatic allocation in DirMaint is not only very useful when combined with skeletons, but also for ad hoc adding of minidisks to existing user IDs. When multiple volumes are grouped together, DirMaint can do rotating allocation on these volumes and distribute the minidisks over the volumes in that group.

Example 9-10 Fragment of EXTENT CONTROL for rotating allocation

```
:REGIONS.
  *RegionId     VolSer     RegStart   RegEnd       Type
  VMLU1R    VMLU1R          001        3338 3390-03     3390 1    USER
  LNX013    LNX013            1       10016 3390-09     3390 1    USER
  LNX014    LNX014            1       10016 3390-09     3390 1    USER
  LNX015    LNX015            1       10016 3390-09     3390 1    USER
:END.
:GROUPS.
  *GroupName RegionList
  ANY  VMLU1R
  LNX (ALLOCATE ROTATING)
  LNX  LNX013 LNX014 LNX015
:END.
```

The (ALLOCATE ROTATING) entry in the LNX group in the example causes DirMaint to distribute the minidisks over these three volumes. When you use automatic allocation in skeleton files, the disks for the new user would otherwise probably be allocated on the same volume. This could be bad for I/O performance if you have a high degree of multiprogramming in the Linux image, especially when you use large disks like (emulated) 3390-9.

9.4.4 Implement exit for minidisk copy

The DVHDXP exit in DirMaint would need to be implemented to allow DirMaint to copy disks that contain a Linux file system. While increasing or decreasing the size of the minidisk with an ext2 file system is fairly hard to do, copying an ext2 file system from one minidisk to the other is not difficult (especially when CMS RESERVEd mini disks are being used, the DFSMS COPY command can to do this).

DirMaint is prepared to use DFSMS when installed. This should allow DATAMOVE at least to copy a CMS RESERVEd minidisk to another extent of the same size and device type. (Unfortunately, we were unable to verify this because DFSMS was not installed on the VM system we used.) We believe DFSMS COPY does not currently copy the IPL records of the disk, so if your Linux images need to IPL from disk (rather than NSS), that would be something to watch out for.

Implementing a routine for DVHDXP using the program shown in Example 10-4 on page 214 is fairly straightforward using the DirMaint documentation. DirMaint development is aware of these restrictions, so it is possible they will be removed in some future version.

9.5 Using an alternate boot volume

Linux for S/390 does not use `lilo` as the Intel implementation does[2]. Being unable to get back to your previous kernel can make testing a new kernel somewhat risky. However, you can mount a small minidisk over /boot and use that as the IPL device; Example 9-11 on page 206 shows how to prepare one of the two IPL minidisks.

Example 9-11 Preparing a separate boot disk

```
# cat /proc/dasd/devices
0205(ECKD) at (94:0) is   dasda:active  at blocksize: 4096, 36000 blocks, 140 MB
0204(ECKD) at (94:4) is   dasdb:active  at blocksize: 4096, 36000 blocks, 140 MB
0201(ECKD) at (94:8) is   dasdc:active  at blocksize: 4096, 468000 blocks, 1828 MB
```

[2] The patches for the 2.4.5 kernel from June 2001 change various things in this area. This may even include an option to select from different kernels.

```
0202(ECKD) at (94:12) is   dasdd:active  at blocksize: 4096, 468000 blocks, 1828 MB
206A(ECKD) at (94:16) is   dasde:active  at blocksize: 4096, 18000 blocks, 70 MB
206B(ECKD) at (94:20) is   dasdf:active  at blocksize: 4096, 18000 blocks, 70 MB
# mke2fs /dev/dasdf1 -b 4096
mke2fs 1.19, 13-Jul-2000 for EXT2 FS 0.5b, 95/08/09
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
18016 inodes, 17997 blocks
899 blocks (5.00%) reserved for the super user
First data block=0
1 block group
32768 blocks per group, 32768 fragments per group
18016 inodes per group
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
# mount /dev/dasdf1 /mnt
# cd /boot
# tar cf - . | tar xf - -C /mnt
# cd /
# umount /mnt
# mount /dev/dasdf1 /boot
# df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/dasdb1             139452     83080     49176  63% /
/dev/dasdc1            1842540    752128    996816  43% /usr
/dev/dasdf1              69720      1812     64312   3% /boot
```

As you can see, the IPL minidisk could have been much smaller than the 100 cylinders we happened to have. This new boot disk can now be updated with the new kernel, and **silo** makes the disk bootable.

Example 9-12 Making the alternate disk bootable

```
# cd /boot
# ls
.   System.map-2.2.16  image             image.config     ipldump.boot
iplfba.boot  parmfile      parmfile.orig
..  boot.map           image.autoconf.h  image.version.h  ipleckd.boot
lost+found   parmfile.map
# cp /usr/src/linux/arch/s390/boot/image .
# cp /usr/src/linux/System.map System.map-2.2.16
# cat parmfile
dasd=0205,0204,0201,202,206a,206b root=/dev/dasdb1 noinitrd
# df .
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/dasdf1              69720      1820     64304   3% /boot
# silo -d /dev/dasdf
o->ipldevice set to /dev/dasdf
```

```
Testlevel is set to 0
IPL device is: '/dev/dasdf'
bootsector is: '/boot/ipleckd.boot'...ok...
bootmap is set to: '/boot/boot.map'...ok...
Kernel image is: '/boot/image'...ok...
original parameterfile is: '/boot/parmfile'...ok...
final parameterfile is: '/boot/parmfile.map'...ok...
ix 0: offset: 00026a count: 0c address: 0x00000000
ix 1: offset: 000277 count: 80 address: 0x0000c000
ix 2: offset: 0002f7 count: 80 address: 0x0008c000
ix 3: offset: 000377 count: 7c address: 0x0010c000
ix 4: offset: 0003ff count: 02 address: 0x00188000
ix 5: offset: 000401 count: 01 address: 0x00008000
Bootmap is in block no: 0x00000402
#
```

To complete the process, you should add the new /boot to /etc/fstab so that the correct System.map will be picked up by **klogd**, but you may want to skip that step the first time your try your new kernel. You can create several alternate boot disks this way, if necessary.

Note: Another approach is described in the IBM Redbook *Linux for S/390 and zSeries: Distributions*, SG24-6264. It renames the /boot directory first, and then creates a new mount point /boot and mounts the new boot disk on that.

Although that approach works as well, we believe the method we detailed in this book is somewhat more elegant.

10

Cloning Linux images

In order to run a large number of Linux images on VM, you need to have those images in the first place. There are different ways to create these images, as discussed in this chapter.

10.1 Overview

One way of creating Linux images is to install each image from scratch, as if you were going with the set of CDs from one server to the next. If you run a significant number of Linux images, and the images are very similar, you may want options that will help you avoid repeating the entire install process for each Linux image. Procedures for this type of repeated installation for discrete servers have been developed for internal use in many organizations.

Often these procedures are tailored to meet the specific requirements of the organization. An initiative for a generic approach is the IBM Open Source project "Linux Utility for cluster Installation" (LUI). The source code for this project is available at the developerWorks Web site. Unfortunately, during this residency we did not have time to see if LUI is applicable to Linux for S/390 as well. Some Linux distributions offer their own approach for repeated installs (such as the YaST shortcuts in SuSE and the `mkkickstart` utility in Red Hat).

When you run Linux images on VM, you have additional options. With VM, you can easily access the disks of the Linux images even when the Linux image is not running. You can use VM utilities to copy the disks from one image over to another image, and then apply the modifications that are needed to use this copy in the next Linux image. This process of making the copy and applying the changes is often referred to as "cloning" Linux images.

10.2 Installing Linux images the easy way

The most straightforward way to install Linux images on a VM system is to create the VM user ID with sufficient minidisks, and install the Linux from some distribution on these minidisks. If you have only a few Linux images on the same VM system, you could simply repeat the process for each of them. This process is well documented in the redbook *IBM @server Linux for zSeries and S/390: Distributions*, SG24-6264.

However, if you have more than just a few Linux images, manually doing the entire installation over and over again is tedious and inefficient, so you may want a few shortcuts. Because the SuSE distribution uses a full-screen menu-based installer, it does not lend itself very well to automated repeated installs. The Turbolinux and Red Hat distribution is easier to automate if you are planning to do a fresh install for each Linux image.

10.2.1 Providing fast access to the install medium

If you have only a small number of Linux images, you may want to do a fresh install for each of them. You may also want to do this for educational or recreational purposes. Rather than getting all the rpm packages via TCP/IP for each install, you can put a copy of the ISO images on a large minidisk and have the Linux image use a R/O link to that disk.

The commands in Example 10-1 show how to mount the ISO images of the installation CDs into your file system before you start the installation program. With the ISO images mounted like this, you can point YaST to the /install/cd1 directory to find the installation material.

Example 10-1 Mount the ISO images via the loop device

```
# mkdir /install
# cd /install
# mkdir cd1 cd2 cd3
# mount -o loop,ro suse-us-s390-cd1.iso cd1
# mount -o loop,ro suse-us-s390-cd2.iso cd2
# mount -o loop,ro suse-us-s390-cd3.iso cd3
# cd /
# yast
...
```

You can simplify this process even further by unpacking the ISO images to a single large ext2 file system (thereby avoiding the use of the loop device during the install). This way you can also add your own packages to the installation medium, or upgrade some of the packages in it.

Example 10-2 Copy the contents of the ISO images to a file system

```
# mount /dev/dasde1 /mnt
# cd /install
# cd cd1 ; tar cf - . | tar xpf - -C /mnt/ ; cd ..
# cd cd2 ; tar cf - . | tar xpf - -C /mnt/ ; cd ..
# cd cd3 ; tar cf - . | tar xpf - -C /mnt/ ; cd ..
# cp /boot/ipleckd.boot .
# cd /
# umount /install
# mount /dev/dasde1 /boot
# ln suse/images/tapeipl.ikr image
# ln suse/images/parmfile parmfile
# ln suse/images/initrd initrd
# silo -d /dev/dasde -r initrd
```

The last few commands in Example 10-2 create links in the root directory of the disk to each of the starter files (kernel, parameter file and initrd image) and a copy of ipleckd.boot. The `silo` command makes the disk bootable for installation.

To install Linux in a new user ID, you can link to this big disk that contains the unpacked three ISO images and IPL it. Installation from such a disk is very fast.

Note: There is a bug in the SuSE install program in that it misses an FBA device (for example, a VDISK) when building the parameter file after installing the packages. After you exit YaST, you must mount the new root device again and edit the parameter file to include your swap device at the beginning of the parameters for the DASD driver.

In order to be able to install further packages after the reboot, you'll want to include the disk with the installation files as well—do that at the end of the parameters for the DASD driver. Finally, run `silo` again.

10.3 Building a quick start disk

By using the "hidden" option of the `silo` command, you can make a disk with a kernel, parameter file, and initrd to be used as the startup system or rescue system (to avoid punching kernel and RAMdisk images). As with the recipe in 9.5, "Using an alternate boot volume" on page 206, you need to mount the disk to be prepared on /boot in order for `silo` to work. The difference is in the `-r` option of `silo`, which allows you to specify the initrd image.

Example 10-3 Using silo to make a quick start disk

```
vmlinux6:/boot # silo -d /dev/dasdh -r initrd.gz
o->ipldevice set to /dev/dasdh
o->ramdisk set to initrd.gz
Testlevel is set to 0
IPL device is: '/dev/dasdh'
bootsector is: '/boot/ipleckd.boot'...ok...
bootmap is set to: '/boot/boot.map'...ok...
Kernel image is: '/boot/image'...ok...
original parameterfile is: '/boot/parmfile'...ok...
final parameterfile is: '/boot/parmfile.map'...ok...
initialramdisk is: 'initrd.gz'...ok...
ix 0: offset: 00016d count: 0c address: 0x00000000
ix 1: offset: 00017a count: 80 address: 0x0000c000
ix 2: offset: 0001fa count: 80 address: 0x0008c000
ix 3: offset: 00027a count: 6d address: 0x0010c000
ix 4: offset: 0002e7 count: 01 address: 0x00008000
ix 5: offset: 001d10 count: 0c address: 0x00800000
ix 6: offset: 001d1d count: 80 address: 0x0080c000
```

```
..[snip].
ix 22: offset: 002520 count: 80 address: 0x0100c000
ix 23: offset: 0025a0 count: 32 address: 0x0108c000
Bootmap is in block no: 0x000002e8
```

You will also notice that the list of blocks in this case is much longer than when no RAMdisk image is specified. If you now **umount** the disk and issue a **sync** command, the disk can be linked from other user IDs to quickly boot a system with RAMdisk.

10.4 Copying disks instead of doing a full install

With virtual machines on VM, you have options other than simply repeating the entire installation process for each image. For example, you can use the DASD Dump and Restore (DDR) utility after a full installation of Linux to copy the contents of the minidisks of one virtual machine to another set of minidisks. This is very practical if you want to keep a copy of your entire Linux system when you are going to do significant changes to your Linux system. That other copy can be started by linking the minidisk at the correct address (this is necessary because the kernel parameters refer to the minidisk address).

Important: Make sure you *shut down* your Linux image before you copy the contents of the minidisks. This is obvious when you run the DDR utility in the same virtual machine that was running Linux. However, when you run the DDR utility in another user ID with a R/O link to your Linux disks, you need to keep this in mind.

This is because when the disks of a *running* Linux image are copied, the resulting file system will at best be "dirty" (not cleanly unmounted, so an **fsck** is required at boot time). Since Linux buffers data in memory before writing it out to disk, your copy could be incomplete and inconsistent if the system is actively doing work. The same applies to normal backups of these disks with your VM backup product.

Instead of switching between IPLs of one of these copies in the same virtual machine, you can also take the copy and IPL it in another virtual machine. However, there are a few complications when you do this. For example, the two copies of Linux you get this way are completely identical, including any configuration changes done by the installation program (e.g. IP address, device addresses, etc).

Therefore, in order to use this approach for generating new Linux images, you need to find the correct point in the installation process to copy the disk. You also need to understand what configuration changes must be repeated for the new image, and how to do these changes.

10.4.1 Copying disks for cloning images

Many will tell you that using DDR is the best way to copy minidisks. While it is probably the cheapest, Table 10-1 shows that it is not always the fastest way to do the job.

Table 10-1 Copying a 250-cylinder minidisk with ext2 file system

	Elapsed time (s)	I/Os	MDC hits	CPU time (s)
DDR copy (first)	30	3014	0	0.148
DDR copy (second)	29	3014	0	0.125
CMS copy (first)	34	13759	41	0.880
CMS copy (second)	19	13759	1166	0.800

The alternative copy on CMS was done with a little pipeline that is shown in Example 10-4 (which we could do because the disk was prepared with RESERVE before we ran **mke2fs** against it). Because this ext2 file system happened to be filled only for 70% in this case, there are still a lot of blocks filled with binary zero. Those blocks do not need to be copied to the target disk. The second run is even faster because MDC is offering a lot of help. DDR does not appear to benefit from MDC.

Example 10-4 Using CMS Pipelines to copy the payload of a reserved disk

```
PIPE
   < tux01a0 tux6mstr j               /* Read from the input disk       */
   | spec number 1.10 ri 1-* n        /* Insert block number in record  */
   | not verify 11-* x00              /* Drop when just '00'X chars     */
   | fileupdate tux01a0 tux60000 k    /* to write them to disk          */

PIPE
     mdiskblk number j 1.2            /* Read boot record from disk     */
   | mdiskblk write k 1.2             /* and write to target disk       */
```

The question remains as to how much you could benefit from MDC in a real-life situation when cloning Linux images, and whether you can spare the CPU cycles. When the file system is more scattered over the disk, or when the file system contains more data than the arbitrary value of 70% used in our test, then the benefit of the CMS-based copy will soon disappear.

10.4.2 Determine the point to copy the disks

All current distributions use a process where you boot Linux with a RAMdisk to populate the disks, and then boot from disk. After booting from disk, SuSE for example runs **SuSEconfig** again and rewrites all kind of files (including the keys for SSH, if you installed that). Therefore, the most practical point at which to copy the disks during the install process is just before this reboot from disk.

Note: This means you must keep this original install system, without booting it, if you want to clone more images from it. If necessary, you can get back into YaST again and add more packages if you have to, and subsequent copies will then also have those packages installed.

10.4.3 Locating the files to be customized for each cloned image

For each cloned Linux image, you now have to customize the files that are prepared during the first phase of the install (with IP address, host name, root password, etc).

There is no easy recipe for this step; determining which files need to be changed will depend on the distribution and the packages installed. This step will have to be done for each new release of the distribution, although the differences between two releases of the same distribution will probably be very small.

Various tools in Linux are available to assist in determining which files to change. (For example, we ran a **find -mtime 0 -type f** command, which showed that only 98 out of 82711 files in our SuSE install were changed). If you look at the list of files, you see there are also a few with "old" and "bak" in their names that you can ignore for this process.

For each file found by this **find** command, you can use **grep** to search for files that contain the IP address or host name, as shown:

```
# grep 192.168.0.2 `find -mtime 0 -type f`
```

We expected the changes for a SuSE distribution to be rather trivial[1] because **SuSEconfig** takes many of the configuration parameters from /etc/rc.config. After the reboot, **SuSEconfig** rewrites most of the files that were identified as changed during the installation.

[1] This turned out to be more complicated than expected because many things happen outside SuSEconfig (for example, /etc/route.conf). The process outlined in the following sections can handle all these changes anyway, so there is less reason to depend on SuSEconfig for some of the work.

When the cloned image uses the same type of network interface and the same DNS and gateway, you do not have to change these when cloning an image. And when you select a unique-enough hostname and IP address for the first install, you may be able to just use a few **sed** commands, as shown in Example 10-5.

Example 10-5 Update the rc.config with IP address and hostname

```
cat /mnt/etc/rc.config \
   | sed s/192.168.0.2/192.168.0.4/ \
   | sed s/tux6mstr/tux60002/ > /root/rc.config
cp /root/rc.config /mnt/etc/rc.config
```

To change the initial root password, you can **chroot** to the file system to be prepared and invoke the **passwd** command. (For SuSE, however, this is somewhat useless since it will prompt for a new password anyway at the first reboot.)

Using diff to find the changes

You can also take a more "brute force" approach to identifying the differences between cloned images by using the **diff** command. The **diff** command compares two files and lists the differences between the files.

For this approach, you perform an identical install according to the book on two different Linux images. Keep these installations the same for the configuration items that will be the same for all your cloned images (e.g. the DNS or domain name), and then deliberately introduce a difference for the items that must be customized for each cloned image (e.g. IP address and hostname).

After completing the installation up to the point where the first boot from disk is required, link both file systems in a single Linux image (the examples here show this running with a RAMdisk system because that's an easier way to configure the DASD driver).

Example 10-6 Mounting the two file systems to be compared

```
# insmod dasd dasd=200,1a0,1a1,1b0,1b1,0cd
Using /lib/modules/2.2.16/block/dasd.o
# cd /mnt
# mkdir a b
# mount /dev/dasdb1 a -r
# mount /dev/dasdc1 a/usr -r
# mount /dev/dasdd1 b -r
# mount /dev/dasde1 b/usr -r
# df
Filesystem          1k-blocks      Used Available Use% Mounted on
/dev/ram2               25901     22377      3524  86% /
/dev/dasdb1            174116     44968    120160  27% /mnt/a
/dev/dasdc1           1415848    692296    651632  52% /mnt/a/usr
```

```
/dev/dasdd1             69628     44968     21068  68% /mnt/b
/dev/dasde1           1415848    692296    651632  52% /mnt/b/usr
```

The fragment in Example 10-6 shows how to load the DASD driver and mount the file systems under the root file system. Because both installs were done on a root file system with two disks, all these disks are needed here.

Unfortunately, these file systems contain absolute path names in the symbolic links, so you can not simply run **diff** against both file trees. Instead, change the absolute path names in the symbolic links to relative ones (there are 69 of those in a "default" install of SuSE). You can identify the problems with the following commands:

```
# find -type l|xargs ls -l $1|cut -b 56-|grep "[^ ]* -> /"
```

Alternatively, if you don't want to change the symbolic links in the file system, you can first run **md5sum** against *each* file in *both* file systems (to compute a checksum for each file), and then use **diff** on the files that have different checksums. Obviously, computing the checksum for each file with **md5sum** is not a cheap process, but on the configuration we used, it completed in two minutes for a "default install" of SuSE. You need to do this once during the preparation.

The scenario in Example 10-7 shows we use the **chroot** command before running **diff**. With **chroot**, you run another command (the shell, in this case) with the specified directory as the root of the file system. The absolute symlinks in the file system now match the file system. The **exit** command terminates the shell and returns back to the configuration before the **chroot** command.

Example 10-7 Computing the checksum for each file in the file system

```
# df
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/ram2               25901     22381      3520  86% /
/dev/dasdb1            174116     48884    116244  30% /mnt/a
/dev/dasdc1           1415848    692296    651632  52% /mnt/a/usr
/dev/dasdd1             69628     44968     21068  68% /mnt/b
/dev/dasde1           1415848    692296    651632  52% /mnt/b/usr
# mount b -o remount,rw
# chroot b
# find / -type f | xargs md5sum $1 > /sums
# sort -k 2 /sums > sumsort
# exit
```

Because **find** did not always return the files in the same order, it was necessary to sort the list on file name. We could have done the sort by piping the output of **xargs** into it, but the temporary files created by **sort** caused a lot of noise in the output.

Using the files with checksums created in the previous steps, we can now identify the files that are different by using **diff** to compare these two files, as follows.

```
diff -u a/sumsort b/sumssort |grep  ^\-  |cut -b 36- \
   | a/usr/bin/gawk '{print "diff -u a"$0" b"$0} ' \
   | grep / |/bin/sh
```

Note: We "cheated" a bit with the **gawk** command. Because the RAMdisk system did not have gawk installed, we used the gawk executable from the mounted file system that we were comparing. However, a better solution would be to install gawk in the RAMdisk system—or to run with a full Linux system.

A portion of the output of running **diff** against these pairs of files is shown in Example 10-8. The **patch** command can take a file like this as its input, to apply the changes to the files in the file system of a cloned Linux image.

Example 10-8 Fragment of the output of diff comparing the files

```
--- a/etc/rc.config     Wed Jul 25 08:42:09 2001
+++ b/etc/rc.config     Wed Jul 25 08:48:21 2001
@@ -132,7 +132,7 @@
 #
 # IP Adresses
 #
-IPADDR_0="192.168.0.2"
+IPADDR_0="192.168.0.3"
 IPADDR_1=""
 IPADDR_2=""
 IPADDR_3=""
```

To use a process like this for customizing cloned images, you have to generate the proper modified version of this output and feed it to the **patch** command. After cleaning up the output of **diff** by hand and removing the patches to files that will be replaced by **SuSEconfig** anyway, a diff file of only 83 lines remained, updating 8 different files in the system.

In the diff file, we replaced things like the host name and IP address by strings that are easy to identify (like :hostname:), and a simple script with a few **sed** commands can now produce the patch file.

Example 10-9 Script to create the specific patch for a cloned image

```
#! /bin/sh
if [ -z $5 ]; then
  echo "Need hostname IP-address gateway-userid gateway-ip gateway-mtu"
  exit
fi
cat generic.diff             \
   | sed "s/:hostname:/$1/  " \
   | sed "s/:cloneip:/$2/   " \
   | sed "s/:gatewayid:/$3/ " \
   | sed "s/:gatewayip:/$4/ " \
   | sed "s/:gatewaymtu:/$5/" \
```

Figure 10-1 illustrates how the first and second install are compared with **`diff`**. The output is then transformed into a generic patch. The shell script in Example 10-1creates a specific patch out of the generic patch. This specific patch is used as the input for the patch program.

Note: If you generate a patch as outlined, you'll also touch files that are marked as "do not modify" because they are maintained by **`SuSEconfig`**.

In theory, you could simply modify rc.config and then run **`SuSEconfig`** to generate the other files. However, in our case, we decided to patch the output files anyway because that avoids running **`SuSEconfig`** and rebooting the system.

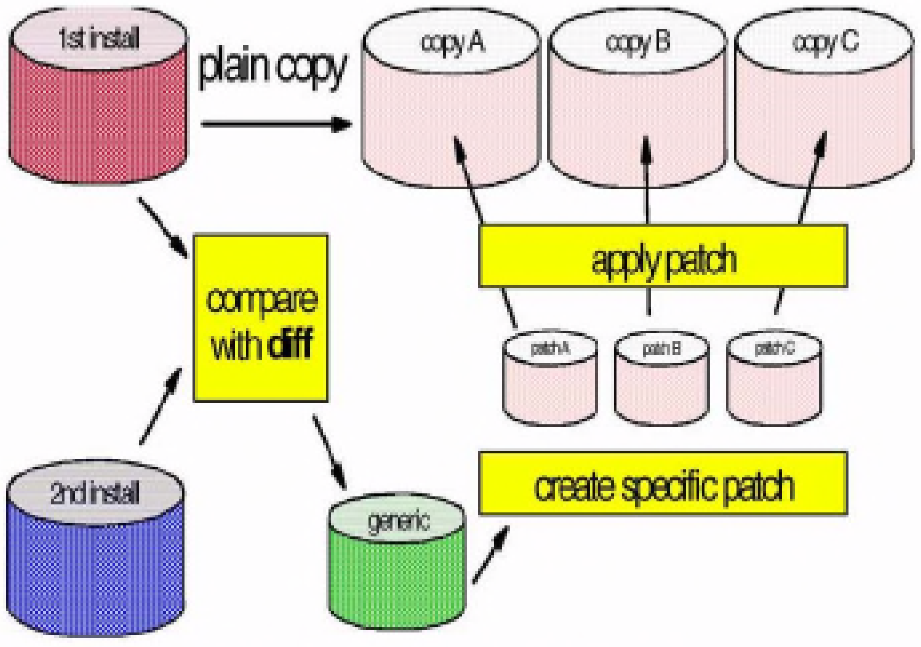


Figure 10-1 Using diff and patch to configure the cloned images

10.4.4 Reducing the number of changes required

Carefully designing your virtual server architecture can significantly reduce the number of changes to be made to each cloned image. If you have a number of different classes of Linux images, then it may be easier to have one separate master image for each category. This will result in fewer changes to be made to a cloned image—which is not only easier to manage, but also *essential* if you want to share disk space among Linux images.

Many useful results can be realized through proper design of VM user IDs for the Linux images. If you keep the differences in the CP directory and the PROFILE EXEC, you can avoid changes to the cloned images. This means, for example, that you keep the same device address for your virtual CTCs on each Linux image, even though they connect to a different address on your virtual hub or VM TCP/IP stack. This way the entry in /etc/modules.conf can be the same for each image.

A significant simplification would be if the cloned images could make use of Dynamic Host Configuration Protocol (DHCP) instead of having hardcoded IP addresses in the configuration files. However, DHCP requires an Ethernet or Token Ring interface, so it doesn't work with point-to-point connections like IUCV and CTC interfaces.