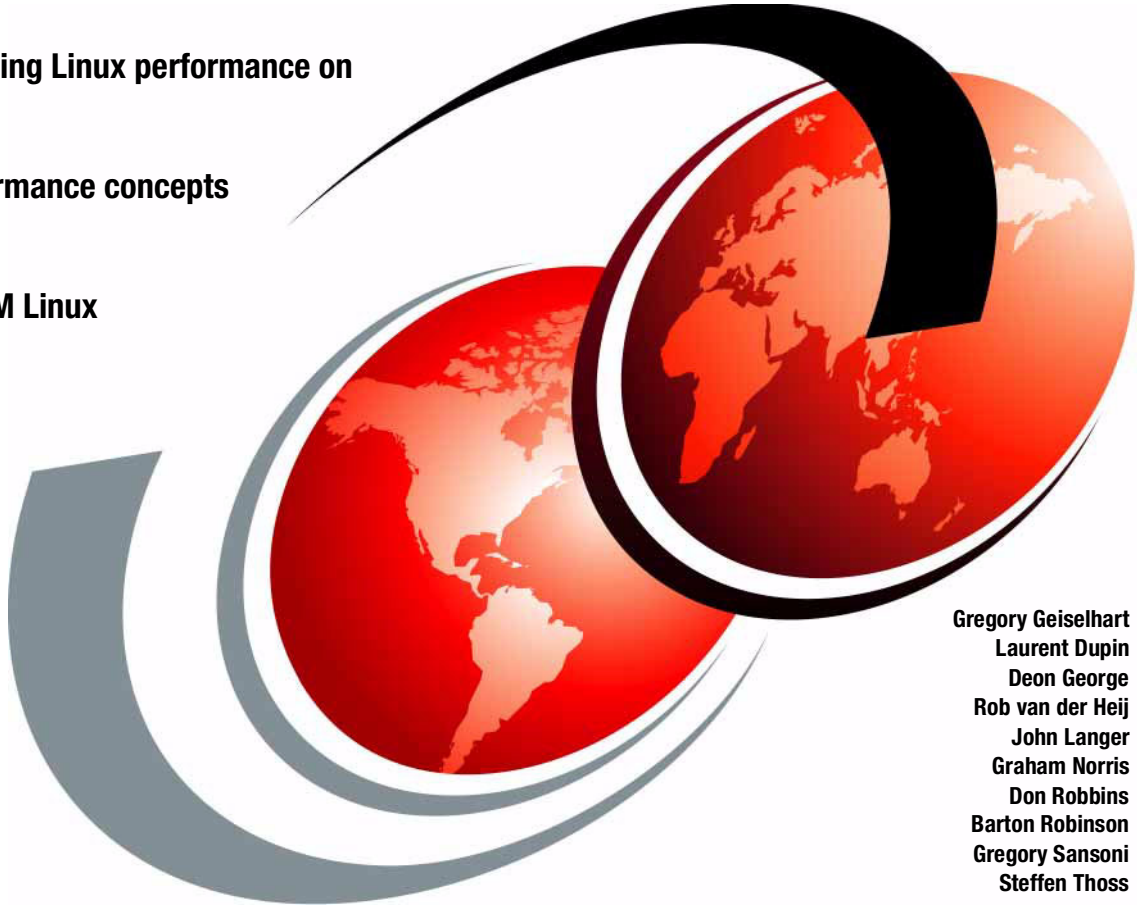
IBM

Linux on IBM @server zSeries and S/390:

Performance Measurement and Tuning

- Understanding Linux performance on zSeries
- z/VM performance concepts
- Tuning z/VM Linux guests



Gregory Geiselhart
Laurent Dupin
Deon George
Rob van der Heij
John Langer
Graham Norris
Don Robbins
Barton Robinson
Gregory Sansoni
Steffen Thoss

Redbooks

ibm.com/redbooks

International Technical Support Organization

Linux on IBM @server zSeries and S/390: Performance Measurement and Tuning

May 2003

SG24-6926-00

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (May 2003)

This edition applies to z/VM Version 4 Release 3 and multiple Linux distributions. Red Hat Version 7.2 for IBM @server zSeries and SuSE Linux Enterprise Server 7 are used for examples in this publication.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

DB2®	eServer™	S/390®
developerWorks®	FICON™	TotalStorage®
DirMaint™	IBM®	VM/ESA®
ECKD™	ibm.com®	WebSphere®
Enterprise Storage Server®	RAMAC®	z/OS®
ESCON®	Redbooks™	zSeries®
@server ™	Redbooks (logo) ™	z/VM®
@server™	RMF™	

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM® Redbook discusses performance measurement and tuning for Linux on IBM @server™ zSeries® and S/390®. It is intended to help system administrators responsible for deployment on Linux for zSeries understand the factors that influence system performance when running Linux as a z/VM® guest.

We consider performance measurement and tuning at both the z/VM and Linux level. Analysis of the memory, processor, DASD, and networking subsystems are provided. Whenever possible, we make tuning recommendations. Measurements are provided to help illustrate what effect tuning controls have on overall system performance.

The system used in this writing the redbook is an IBM @server zSeries 900 (z900) running z/VM Version 4.3 in an LPAR. The Linux distributions used in this redbook include Red Hat Version 7.2 for zSeries (based on a Linux 2.4.9 kernel) and SuSE SLES7 (based on a Linux 2.4.7 kernel). With the exception of 4.2, “Exploiting the shared kernel” on page 37, the examples in this redbook use the Linux kernel as shipped by the distributor.

The z900 is configured for:

Main memory	3 GB
Expanded storage	256 MB
Minidisc cache (MDC)	512 MB
Total LPARs	15
Processors	Two Integrated Facilities for Linux (IFLs) dedicated to the logical partition (LPAR)

The direct access storage device (DASD) storage used in producing this redbook are RAMAC® Virtual Array (RVA) units.

The intent of this redbook is to provide guidance on measuring and optimizing performance using an existing zSeries configuration. The examples are intended to demonstrate how to make effective use of your zSeries investment. The workloads used are chosen to exercise a specific subsystem; any measurements provided should not be construed as a benchmark.

In developing this redbook, we use two performance monitoring tools designed for z/VM and z/VM Linux guests:

- ESALPS by Velocity Software
- OMEGAMON for VM by Candle Corporation

Similar performance monitoring function is available from IBM using the IBM Performance Toolkit for VM (Virtual Machine), an optional priced feature of z/VM 4.4 (5739-A03). This program can be used to monitor VM system performance and analyze bottlenecks. It can be used as a real-time performance monitor and also for analysis of history and trend files of accumulated performance data. As a real-time monitor, the toolkit provides displays of VM CPU, LPAR, channel, and I/O performance, as well as resource consumption, response times, and communication rates. When installed and configured for Linux monitoring (see Appendix C), the Performance Toolkit for VM can report Linux-specific performance data, including Apache request rates and size, and also Linux CPU utilization, Linux memory utilization, Linux network activity, and Linux file system usage.

ESALPS overview

ESALPS, the Linux Performance Suite, is a suite of products provided by Velocity Software. The products that make up the suite include:

- **ESAMAP**
  The VM Monitor Analysis Program, providing performance reports on all aspects of VM/ESA® and z/VM performance.
- **ESAMON**
  The VM Real Time Monitor, providing real-time analysis of performance.
- **ESATCP**
  The network and Linux data collection program.
- **ESAWEB**
  A very fast VM-based Web server.

In addition to the four products, ESALPS provides a Web-based interface to view performance data through a Web browser and many control facilities.

Monitoring requirements

There are many requirements for data collection met by ESALPS. Data is provided for:

- **Capacity planning**
  Long term data in the form of a performance database (PDB) is needed as input to long term capacity planning and trend analysis. Full historical data functions are provided with collection and many forms of data extraction tools.
- **Performance analysis**
  Trend data enables an analyst to detect performance changes in any of thousands of potential problem areas. The performance database allows analysts to determine what of many potential changes occurred in the system. Reporting on specific periods of time can be performed, enabling an in-depth performance analysis of performance problems.
- **Real-time performance**
  Beyond the traditional “entry level” real-time performance reporting of the top users and system utilization, real-time performance analysis is provided for all subsystems, user activity, and Linux (and many other platforms) servers. Network data is also provided real time.
- **Linux data**
  With the advent of virtual Linux server farms on z/VM, performance data is required.

Standard interfaces

ESALPS uses standard interfaces for all data collection. The advantage to using the standard interfaces provided is that when there are a multitude of releases and distributions available, the standard interfaces provide consistent data sources. Supported interfaces include:

- z/VM provides a “monitor interface” that has been available since 1988. Since then, this interface has provided a consistent view of performance of VM systems.
- Network performance is collected using simple network management protocol (SNMP), the standard for network management.
- NETSNMP, an open source software package, provides host data for Linux and other platforms.
- VM application data interface is used by applications to insert data into the monitor stream consistent with the standard monitor interface. ESATCP uses this interface to ensure consistent data collection that allows full integration of Linux and VM data.

Performance database

ESALPS provides both real-time data and historical data for in-depth analysis. The performance data is collected daily with a one minute granularity based on the monitor interval. A longer term archive is collected, usually with a granularity of 15 minutes. This performance database (PDB) includes VM data, Linux data, and network data.

Real-time monitoring with ESAMON

ESAMON uses:

- Historical reporting
- Linux reporting
- Network reporting

Velocity Software has been in business since 1988 supporting the VM environment. With a focus on VM, and now z/VM, Velocity Software added TCP/IP network analysis and then Linux, Microsoft Windows NT, Sun, and other platforms to the product data collection facilities.

OMEGAMON for VM

OMEGAMON for VM is a monitor for the z/VM operating system including 31- and 64-bit z/VM images. It includes an interactive component, a background collection component, and a historical reporting component called EPILOG.

The interactive OMEGAMON component consists of a 3270-based menu system displaying current data for many major areas of interest, further menus displaying historical data, many individual commands, the ability to issue VM commands, the ability to take actions when events occur, and the ability to create custom menus and displays. Also included is degradation analysis, which analyzes the reasons the system, a virtual machine, or a group of virtual machines is degraded and graphically displays the reasons.

The background collector is intended to run at all times. Its primary function is to record data to DASD for later historical reporting, but it also collects some data used by the interactive OMEGAMON user interface. Included with the collector are tools to enable archiving or combining data, or both, to tape or disk.

EPILOG consists of many predefined reports and graphs, including bottleneck reports, the ability to create custom reports, and the ability to export data in a format suitable for inclusion in spreadsheets or other data reduction or analysis tools.

OMEGAMON XE for Linux

OMEGAMON XE for Linux is one of a family of Candle monitoring tools that share a common infrastructure and user interface. It reports current data interactively and includes a historical data collection facility.

The interactive user interface of OMEGAMON XE runs on a Windows workstation and shows information for many monitored systems (which can be any of the OMEGAMON XE family, not just Linux) in one place. (Data from multiple systems can be combined using Candle's related OMEGAMON DE product.) The user interface consists of multiple workspaces, which are grouped in several predefined sets, or which can be user defined. OMEGAMON XE can take action or present advice, or both, when predefined or user-specified situations occur. Data can be displayed as annotated tables, charts, gauges, and so forth. Links from one workspace to another exist in the predefined workspaces and can be created by a user as required.

OMEGAMON XE can create historical data for many of the data attributes it collects. The user selects what data is to be recorded and how frequently. Historical data can be collected either on each individual system, or centrally, and can be periodically warehoused in a central database.

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.

Gregory Geiselhart is a Project Leader for Linux on zSeries at the International Technical Support Organization, Poughkeepsie Center.

Laurent Dupin has worked on z/VM and Linux performance since he joined the EMEA zSeries Benchmark Center in Montpellier, France. He previously worked for 10 years as a z/OS® and Sysplex specialist for IBM Global Services, and became interested in Linux for zSeries during a two-year assignment at the Boeblingen Lab in 1999.

Deon George is part of the IBM Software Group in Australia and has worked for IBM for five years. He has been working with Linux for 10 years and with Linux on S/390 for one year. He is a strong believer that Linux will be successful in the enterprise and will work hard to make sure that any technology solution can include Linux and also be successful.

Rob van der Heij is a z/VM and Linux Systems Programmer with IBM in the Netherlands. He has 20 years of experience with VM, but is still pretty much a Linux newbie. His professional interest includes most that runs on z/VM, in particular Linux. He is a regular speaker at IBM Technical Conferences and participates in Linux and VM mailing lists.

John Langer is an Advisory Software Engineer from IBM Poughkeepsie, working in the e-Business Performance group for z/OS Solution Evaluation Test. He has been with IBM for 25 years, and has been involved with performance testing for the last seven years. John has a degree in geology from the State University of New York.

Graham Norris is an Englishman working as an Advisory Engineer for Candle Corporation in California. He has 31 years of experience with IBM mainframes and has also worked with many other systems, IBM and non-IBM, big and small, during that time, but minimal Linux. His areas of expertise include system automation, capacity planning, and performance measurement.

Don Robbins is a System Engineer for Velocity Software in the U.S. He has 18 years of experience in the VM, UNIX, and Linux fields. His areas of expertise include being a system abuser on any platform that he has worked on, and as a result, performance has been a very important part of his life.

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, the *VM/ESA Tuning Guide* published by Velocity Software, and *Linux on IBM @server zSeries and S/390: ISP/ASP Solutions*, SG24-6299. He is the author and developer of ESAMAP and ESATCP.

Gregory Sansoni is a Staff Software Engineer, working at IBM in Poughkeepsie. He has five years of experience with zSeries performance testing and analysis, with the past two years devoted to Linux on z/VM scalability and performance topics. He holds a degree in Electrical Engineering from Penn State. His areas of expertise include ornithology, with a concentration in the study of species Eudyptula minor.

Steffen Thoss is a Linux for zSeries performance analyst in Germany. He has four years of experience in the Linux for zSeries field. He holds a degree in Computing Science from Berufsakademie Sachsen Dresden. His areas of expertise include zSeries networking and Linux for zSeries performance. Currently, he is working in the Linux for zSeries performance team at the IBM Laboratory Boeblingen, Germany.

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

Terry Barthel, Dave Bennin, Alison Chandler, Roy Costa, Al Schwab, Bill White
International Technical Support Organization, Poughkeepsie Center

Bob Haimowitz
International Technical Support Organization, Raleigh Center

Erich Amrehn, Utz Bacher, Klaus Bergmann, Dr. Juergen Doelle, Mario Held, Martin Kammerer, Jens Osterkamp, Stefan Stahl, Holger Wolf
IBM Germany

Bill Bitner, Bill Cosnett, Dennis Musselwhite, Brian Wade
IBM Endicott

Rick Tarcza
IBM Poughkeepsie

Michel van Kruistum
IBM Netherlands

John P. Hartmann
IBM Denmark

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our 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:

  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

Virtualization and server consolidation

In this chapter, we discuss how to lower overall operating costs by running Linux as a z/VM guest. We examine sharing real hardware resources using the virtualization technology provided by z/VM. Then, we consider what can benefits can be expected from performance tuning.

1.1 Server consolidation and virtualization

If we compare the raw cycle speed of a zSeries CPU with an average Pentium processor, it is not hard to think that zSeries would have a disadvantage. In this redbook, we use as an example an IBM WebSphere® workload generated by the Trade 2 Benchmark. If that workload uses the full 100% of a modern PC for 24 hours a day, the same workload would easily use an entire zSeries CPU for the whole day. Even though a zSeries machine has multiple CPUs, there will be many situations where it still is not economical to host such a workload on zSeries hardware. Fortunately, there are many other situations where using zSeries *does* make a lot of sense.

Consider the case where the workload does not use the machine for the full 24 hours, but only for 12 hours. And imagine there is another similar workload using the same amount of resources, but during the other 12 hours of the day. In that situation, we can put both Linux systems on the same zSeries machine and let them use the CPU cycles they need. When multiple Linux systems run on the same zSeries machine, each Linux system is made to believe it has dedicated access to a defined portion of the zSeries machine. The zSeries hardware and z/VM take care of the smoke and mirrors without actually dedicating portions to that Linux system.

Each Linux system runs in its own *virtual machine*. The characteristics of that virtual machine (memory size, number of CPUs) define the hardware that Linux sees. The tuning controls in z/VM specify how real hardware resources are allocated to the virtual machine.

1.1.1 Virtualization of the CPU

CPU virtualization is accomplished by timesharing. Each Linux guest in turn gains access to a CPU for a period of time. After that, the real CPU is free to run the work of the other Linux system.

The cost of running both of these workloads on discrete servers is twice the cost for running only one of them, but with zSeries, we run both workloads for the price of one. Realistic business applications often even run far less than 50% of the time, so that allows you to run even more of these workloads on the same zSeries machine. The workload from these business applications is most likely also not in a single burst per day, but in multiple short ones over the day. Given a large enough number of servers and short enough intervals of workload, the chances of spreading the total CPU requirements over the full day become better.

Because the zSeries hardware is specifically designed to do timesharing, z/VM can switch between tasks in a very cost effective manner.

1.1.2 Virtualization of memory

In addition to the CPU requirements discussed, the workload also has memory requirements. The amount of memory needed for the Linux system to run is the working set size. Virtualization of memory on z/VM is done through paging. The Linux systems take turns using the main memory of the machine. Paging volumes and expanded storage are used to hold the pages of the inactive Linux systems so that they can be brought in by z/VM when needed.

The challenge for z/VM is to be able to bring in the working set of a Linux system quickly enough that no time is wasted when a Linux system has work to do.

1.1.3 Levels of virtualization

The virtual machine provided by z/VM to run the Linux system is not the only virtualization that is done. The Linux system itself runs multiple processes, and the operating system allocates resources to each of these processes to allow them to get their work done. Some of the processes running on Linux run multiple tasks and allocate their resources to those tasks. And if the z/VM system runs in an logical partition (LPAR), z/VM also uses only part of the real hardware.

These multiple layers of virtualization make it hard for an operating system to find the best way to use the allocated resources. In many cases, tuning controls (knobs) are available guide the operating system in this. Especially when the operating system was not designed to run in a shared environment, some of the knobs turn out to have surprising negative side effects.

1.2 Sharing resources

When we say a resource is shared, this can have different meanings:

- **Multiple virtual machines take turns using a resource.**
  This is most obvious with sharing the CPU. z/VM dispatches virtual machines one after the other on the real processor; each virtual machine believes it “owns” the processor during its time slice. Main memory is also shared as private pages in the working set of each virtual machine are brought in and out of main memory as needed. Again, z/VM creates the illusion that a shared resource is owned by a virtual machine. This type of sharing is not free; there is an overhead in z/VM for switching back and forth between virtual machines. This overhead increases as more virtual machines compete for the CPU.

- **z/VM allows virtual machines to really share memory pages.**
  In this case, a portion of the virtual memory of the Linux virtual machine is mapped in such a way that multiple virtual machines point to the same page in real memory. In z/VM, this is done through named saved systems (NSS). It is possible to load the Linux kernel in an NSS and have each Linux virtual machine refer to those shared pages in memory, rather than require them to have that code in their private working set. There is no additional cost for z/VM when more virtual machines start to share the NSS. Note that not all memory sharing is through NSS. The virtual machines in z/VM also compete for main memory to hold their private working set. The NSS also holds only a small part of the working set of the Linux virtual machine.

1.2.1 Overcommitting resources

It is possible to overcommit your hardware resources. When you run eight virtual machines with a virtual machine size of 512 MB in a 2 GB z/VM system, you overcommit memory roughly with a factor of two. This works as long as the virtual machines do not require the allocated virtual storage at the same time.

Overcommitting resources is a good thing; we also do this in normal life. A restaurant, for example, could be seating 100 people and allow every customer to use the restrooms. The service can be provided with only a small number of restrooms. The same applies to a cinema, but that needs more restrooms per 100 people because of the expected usage pattern. This shows how the “workload” affects the ability to share a hardware resource. This example also shows that partitioning your resources (separate restrooms for male and female guests) reduces your capacity if the ratio between the two different workloads is not constant. Other requirements such as service levels might require you to do so anyway.

When the contention on the shared resources gets higher, chances of queueing get larger. This is a consequence of sharing that can not be avoided. When a queue forms, the requesters are delayed in their access to the shared resources. Whether such a delay is acceptable depends on service levels and other choices you make.

Overcommitting is necessary to share a resource. Big problems normally arise when *all* resources in the system are being overcommitted.

1.2.2 Top speed versus mileage

Benchmarks traditionally deal with measuring the maximum throughput of an application. You will see references to the maximum number of transactions per second, the number of floating point operations per second, and the number of megabytes transferred per second. The maximum throughput of the system is

seldom a good indication of how well a business application runs. The main reason for measuring the maximum throughput is that it is relatively easy to do the measurements.

With multiple virtual machines on z/VM competing for resources, we care much more about the efficiency of the application. The metric for such a measurement is a relation between resources, such as megabytes transferred per CPU-second, or the number of CPU-seconds per transaction.

1.3 The art of tuning a system

A z/VM system offers many controls (tuning knobs) to influence the way resources are allocated to virtual machines. Very few of these controls in z/VM increase the amount of resources available. In most cases, the best that can be done is take away resources from one virtual machine and allocate them to another one where they are better used. Whether it is wise to take resources away from one virtual machine and give them to another normally depends on the workload of these virtual machines and the importance of that work.

Tuning rarely is a “one size fits all” approach. Instead, you will find that a system tuned for one type of workload performs poorly with another type of workload. This means you must understand the workload you want to run and be prepared to review your tuning when the workload changes.

1.3.1 What tuning does not do

It is important to understand you can not run more work than you can fit in the machine. If your zSeries machine has two CPUs and the workload you want to run consists of three Linux virtual machines running WebSphere where each of them runs a CPU for 100% all day, then it just will not fit. There is no z/VM tuning that will make it fit (but there might be issues with the application to make it use less than 100% all day).

When a zSeries machine has four CPUs and the workload consists of three Linux virtual machines that each use a CPU for 100% all day, there is little to tune. In this case, z/VM has sufficient resources to give each Linux virtual machine what it requires (though one might want to look into changes to the configuration that allow you to use all four CPUs and make things run faster).

1.3.2 Where tuning can help

So even when the different workloads add up to less than the total amount of resources available, you might find you are still unable to run the workload. The

reason for that might be that the system is short on another resource. In such a situation, proper tuning can make a difference.

Before tuning the system and workload, you need to understand what resource is the limiting factor in your configuration. Tuning changes tend to fall into one of these categories:

- **Use less constrained resources**
  With a memory-constrained system, one option might be to reduce overall z/VM memory usage by reducing the virtual machine size of Linux guests.
- **Get a larger share of a constrained resource**
  In the case of a memory-constrained system, this can mean reserving some memory pages for one particular Linux virtual machine, at the expense of all others.
- **Increase total available resources**
  The most obvious approach is to buy more hardware. However, additional resources can be made available by stopping unneeded utility services.

Be aware that tuning does not increase the total amount of system resources. It simply allows those resources to be used more effectively.

1.3.3 Exchange of resources

You can view tuning as the process of exchanging one resource for another. Changes to the configuration make an application use less of one resource, but more of the other. If you consider IT budget and staff hours as a resource as well, even the purchase of additional CPUs is an exchange of one resource for the other. When you tune your configuration to less of one resource, it should not be a surprise that it will use more of another resource.

Consider for example the WebSphere Performance Benchmark Sample workload (discussed in Appendix A, “WebSphere Performance Benchmark Sample workload” on page 155). This is a three-tier configuration with a front-end Web server, a WebSphere application, and a database back-end. You can choose to run each tier in its own virtual machine or run them all in the same virtual machine. Obviously, running three virtual machines increases some costs, because as you duplicate the Linux operating system and infrastructure, you have the additional cost of the communication between the virtual machines, and you duplicate items that otherwise could have been shared. However, by using three virtual machines, you can tune the resources given to each of these virtual machines. You can make the Web server very small so that it can be brought into memory easily and will run quickly. You can make the database virtual machine larger so that it can cache a lot of the data and thus avoid some of the I/O to disk. By using three different virtual machines, z/VM can dispatch them on real CPUs independently. So the WebSphere server might be able to use more CPU cycles

because it does not have to wait for the storage of the database server to be brought in.

By just recording the number of transactions per second on an unconstrained system, you will never see these subtle details. And indeed, in an unconstrained environment, it would be best to make everything as big as you can and drive it as hard as you can. In real life however, very few of us can afford an unconstrained environment.

1.3.4 Workload profile

Real business applications have a workload profile that varies over time. The simplest form of this is a server that shows one or more peaks during the day. A more complicated workload profile is when the application is CPU intensive during part of the day, and I/O intensive during another part.

The most cost efficient approach to run many of these types of applications would be if we could adjust the capacity of the server during the day. While this might appear unrealistic, this actually is what we want z/VM to do. Portions of the virtual machine are brought into main memory to run. Inactive virtual machines are moved to paging space to make room.

2

z/VM memory and storage concepts

This chapter introduces the z/VM memory and storage subsystem. Topics include:

- The z/VM storage hierarchy
- Guidelines for allocation of z/VM storage
- z/VM use of memory
- Virtual memory as seen by Linux guests
- Influencing z/VM memory management
- Paging and spooling

2.1 The z/VM storage hierarchy

Both z/VM and Linux use what is known as virtual storage, or virtual memory. Because “storage” is also used to refer to media other than memory, this section uses the term “memory” to refer to electronic read-write memory, also known as RAM, in which programs and its data are kept while a program is running. Note that z/VM and Linux use different terms for the same thing: z/VM normally refers to memory as storage.

Briefly, virtual memory is a method of allowing more programs and data to share real (physical) memory at the same time than would otherwise be the case. At any given moment, a program can only be accessing a very small amount of memory; even over several seconds, a program is highly unlikely to access more than a fraction of the total memory it has assigned to it. Virtual memory systems use a mechanism called paging that tries to ensure that memory that is actively being used is in real memory, and memory that is not being actively used is temporarily saved to disk, and the real memory made available for other memory that is actively in use. In z/VM and Linux, memory is managed in 4 K pages.

The z/VM storage hierarchy uses three types of memory:

- **Main memory**
  Often referred to as *main storage*, this memory is directly accessible by user programs. Programs execute in main memory. All I/O operations occur within main memory. The size of main memory is limited to the amount physical memory.
- **Expanded storage**
  Expanded storage exists in physical memory, but is addressable only as whole pages. Physical memory allocated as expanded storage reduces the size of the main memory. Expanded storage is optional, and its size is configurable.

  Expanded storage acts as a fast paging device. As demand for main memory increases, z/VM can page to expanded storage. Because it exists in physical memory, paging to expanded storage can be much faster than paging to direct access storage device (DASD).

  Note: Expanded storage can also be used for minidisk cache (MDC).

- **Paging space**
  Paging space resides on DASD. When paging demands exceed the capacity of expanded storage, z/VM uses paging space.

The relationship between the types of z/VM storage is depicted in Figure 2-1 on page 11.

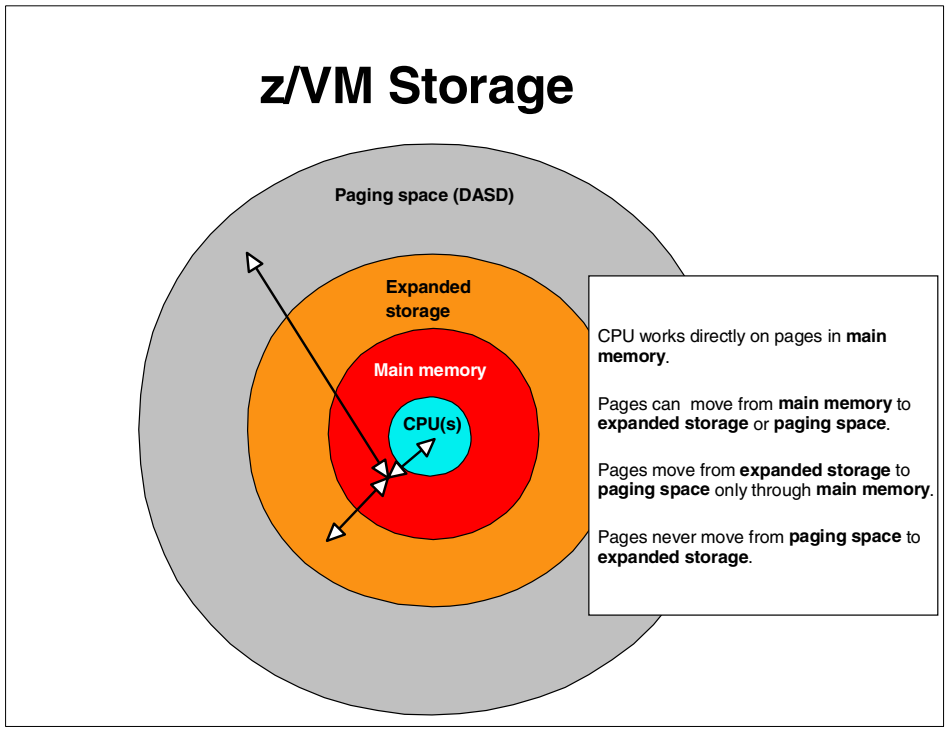


Figure 2-1 The relationship between the types of z/VM storage

The combination of main memory, expanded storage, and paging forms the z/VM virtual memory address space. As illustrated in Figure 2-1, pages move within z/VM virtual memory to accommodate demand:

- **z/VM guests run in main memory.**
  When dispatched, guests execute in main memory. Not all guest pages need to reside in main memory when running. An inactive page can reside in expanded storage or in paging space, or in both.
- **Paging occurs between main memory and expanded storage.**
  As demand for main memory increases, z/VM might move inactive guest pages to expanded storage. As those pages become active, z/VM moves them back to main memory.
- **Paging also occurs between main memory and paging space.**
  If no expanded storage is configured, z/VM pages between main memory and paging space. If the paging demand exceeds the expanded storage capacity, z/VM pages to and from main memory and paging space.

- **Pages do not move directly between expanded storage and paging space.**
  Pages that move from expanded storage to paging space must first be brought to main memory.

2.2 Guidelines for allocation of z/VM storage

As discussed in 1.2.1, “Overcommitting resources” on page 4, overcommitted memory is a normal and desirable situation on z/VM. Memory overcommitment allows z/VM to provide more total server utilization (and therefore, a lower overall cost). The z/VM storage hierarchy is designed to optimize paging on an overcommitted system.

With 64-bit support, the question arises of whether there is a need for expanded storage (why not configure all physical memory as main memory instead). The general recommendation is to configure z/VM with expanded storage.

Expanded storage often results in more consistent or better response time. Factors that suggest expanded storage improves response time include:

- **Paging will likely occur in a z/VM system.**
  The logic for allocating all physical memory as main memory is that paging only occurs if there is a shortage of main memory, and therefore, expanded storage only increases this possibility, However, overcommitment of memory on z/VM is a *normal* and *healthy* practice. Therefore, it is better to prepare for paging than attempt to prevent it.
- **z/VM paging algorithms are tuned for expanded storage.**
  Moving pages from expanded storage to paging storage is much more efficient than moving pages from main memory to paging storage.
- **Parts of CP and its control blocks must reside below 2 GB.**
  Even with 64-bit support, parts of CP must still reside below the 2 GB line. Guest pages being referenced by CP for some operations (such as I/O) must reside below 2 GB. This can create contention for storage below 2 GB. Contention below 2 GB can be identified by observing paging-to-paging space at times when main memory above 2 GB is unused.

Note: When contention for memory below 2 GB is heavy, allocating 2 to 3 GB of expanded storage might help. For most systems however, 2 to 3 GB of expanded storage is probably excessive.

As a general estimate, start with expanded storage configured to be 25% of the physical memory allocated to z/VM. Systems with low contention below 2 GB can reduce this ratio. For more tips about storage configuration, see the VM Performance Tips page at:

http://www.vm.ibm.com/perf/tips/storconf.html

Note: The part of VM that runs everything else is called CP, originally from the term Control Program. Wherever you see “CP,” it refers to the core operating system part of z/ VM.

2.3 z/VM use of memory

Example 2-1 on page 14 shows a typical system storage map for a 64-bit z/VM system.

Example 2-1 z/VM real storage map

```
_________________ ZSMAP    CMS     OM/VM    V610.99 CMS  02/24/03 09:49:27    B
> OMEGAMON for VM - Show layout of z/VM's real storage
> S.C          Help PF1           Back PF3          Up PF7          Down PF8
================================================================================
>    REAL AND VIRTUAL STORAGE:  Enter a selection letter on the top line.

>Primary:  A-DPA           B-Real storage   *-SYS storage    D-Paging   E-VDISK
>Expanded: F-System info   G-Block paging   H-Page info      I-Paging   J-VDISK
>General:  K-Shared segs   L-Frame table    M-Free by user   N-Free map
================================================================================
>                            SYSTEM STORAGE MAP

 SYS   >> z/VM   V4 R3.0  SLU 0201
+         CPU: 2064 #0C0ECB-1C0ECB
 SMAP   Major Area    Size Minor Area    Size           Address Range
+       --------------------------------------------------------------------
+       Dynamic Area 1024M Dynam Paging 1024M 000000007FFFF000-00000000BFFFFFFF
+       --------------------------------------------------------------------
+       Fixed Stg      12M CP Frame Tbl   12M 000000007F3FF000-000000007FFFEFFF
+       --------------------------------------------------------------------
+       Dynamic Area 2031M Dynam Paging   36K 000000007F3F6000-000000007F3FEFFF
+                          Trace Tbl 00  400K 000000007F392000-000000007F3F5FFF
+                          Dynam Paging   36M 000000007CF98000-000000007F391FFF
+                          Trace Tbl 01  300K 000000007CF4D000-000000007CF97FFF
+                          Dynam Paging 1994M 0000000000539000-000000007CF4CFFF
+       --------------------------------------------------------------------
+       Fixed Stg     216K Chan Measure  216K 0000000000503000-0000000000538FFF
+       --------------------------------------------------------------------
+       Dynamic Area 2352K Dynam Paging 2352K 00000000002B7000-0000000000502FFF
+       --------------------------------------------------------------------
+       Fixed Stg    2780K CP Nucleus   2772K 0000000000002000-00000000002B6FFF
+                          Prefix        8192 0000000000000000-0000000000001FFF
+       ====================================================================
+       Total online 3072M
+       ====================================================================
```

The areas of memory labelled `Dynam Paging` are the dynamic paging area (DPA), the area memory used by VM for running guest operating systems such as Linux. The other areas are used by VM itself.

Memory pages in the DPA are eligible to be temporarily stored elsewhere when not actively used. If a page of memory is not located in DPA when required, a *demand page-in* operation is initiated. If no free page exists in the DPA, VM first pages out an inactive page. For efficiency, z/VM attempts to keep a pool of free memory to reduce the number of demand page-outs (it is more efficient to perform a sweep through memory and move several pages out at once).

Whenever a virtual machine moves from the dispatch list to the eligible list, its memory usage is examined. At that time, unused memory pages are trimmed from the virtual machine to reduce its memory footprint.

As shown in Example 2-1 on page 14, storage above 2 GB (0x7FFFFFFF) is entirely DPA. CP stores its control blocks and data below 2 GB. Do not assume that this situation will remain unchanged in future releases of z/VM, because limiting CP control blocks to addresses below 2 GB limits the abilities of z/VM. For example, the frame table for 512 GB of real memory would require the entire lower 2 GB, leaving no room for CP itself.

2.4 Virtual memory as seen by Linux guests

Virtual memory refers to the combination of main memory, expanded storage, and paging space. Linux guests running under z/VM see memory as a contiguous area extending from a low address of zero to a high address equal to the virtual machine size. However, these pages might not be contiguous in z/VM virtual memory, and might moved to accommodate demand. In fact, z/VM might move a Linux guest's memory pages out of main memory without notifying the guest.

2.4.1 The double paging effect

z/VM memory management can lead to a situation referred to as *double paging*, an effect illustrated in Figure 2-2 on page 16.

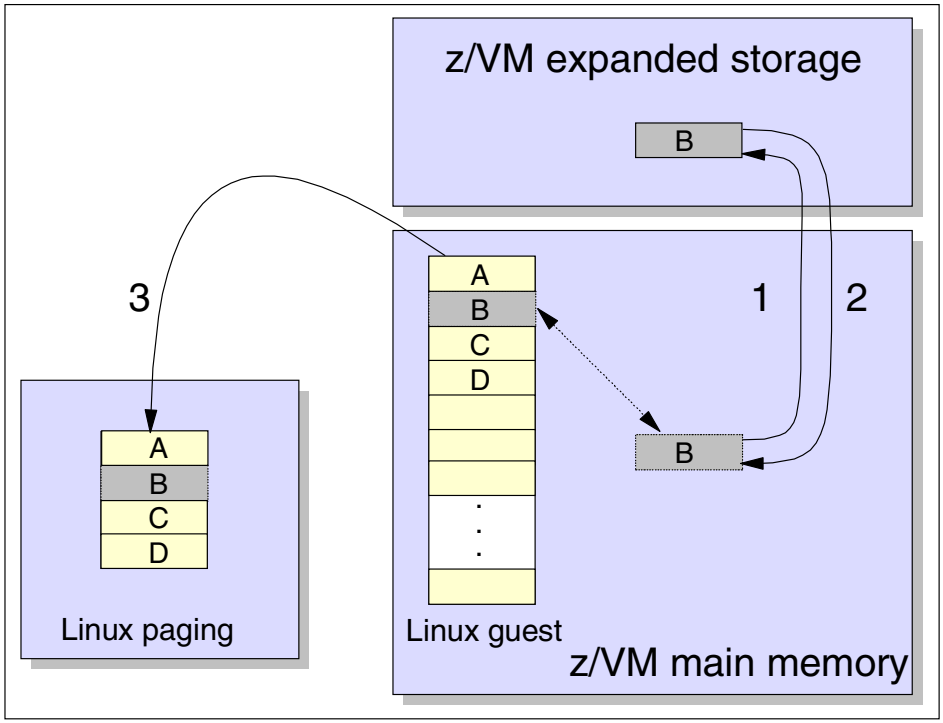


Figure 2-2 Illustrating the double paging effect

Figure 2-2 depicts memory pages used by a Linux guest. The event sequence leading to double paging is denoted by the numbered arrows:

1. **Memory page B is paged out from z/VM main memory.**
   This memory page is mapped to a running Linux guest. After a period of inactivity, z/VM moves the page to expanded storage.

   Note: In this example, we show the page-out occurring to expanded storage. If no expanded storage was available, the page-out would occur to paging space, an even more expensive operation.

   The page is now available for demand page-in. z/VM has stolen the page, but the Linux guest is unaware that the page-out occurred.

2. **Memory page B is paged back into z/VM main memory in response to a page-out attempt by the Linux guest.**
   To alleviate a memory constraint, Linux identifies inactive pages (including page B) to move to its swap device. The Linux guest page faults when attempting to access page B. This, in turn, causes z/VM to page-in memory page B to satisfy that request.
3. **Linux completes its page-out attempt by moving page B to its swap device.**
   After page B is paged back into main memory, Linux pages it out to its swap device. In the end, z/VM has moved pages back into main memory simply to allow Linux to move it out again.

Double page faults are not unique to Linux; z/OS running under z/VM can experience the same effect. It is a result of two parties attempting to manage memory. The solution is to ensure one party does not attempt to page:

- **Make the Linux guest virtual machine size small enough for z/VM to keep in main memory.**
  Double paging might still occur if many smaller Linux guests compete for z/VM main memory.
- **Make the virtual machine size large enough that Linux will not attempt to swap.**
  This can lead z/VM to frequently page fault. This can be improved by PAGEX/PFAULT, an asynchronous mechanism that allows z/VM to inform Linux that a requested memory page is not resident in z/VM main memory. On receiving this notification, Linux will attempt to dispatch another process.

2.4.2 Allocating memory to z/VM guests

z/VM virtual machines are assigned memory in the z/VM directory entry. Two values in the directory entry specify memory for each user:

- First is the size of the guest's virtual machine when it logs on.
- Second is the maximum size to which the virtual machine can change itself after it has logged on.

If a guest changes its virtual machine size, the memory is reset, and the initial program load (IPL) occurs again. As a result, changing a virtual machine size is something usually done by a human user, not by an operating system running in a virtual machine.

Table 2-1 Storage allocation activities

Activity	Command or directory entry
Initial storage allocation	USER directory entry
Maximum storage allowed	USER directory entry
Change storage allocation	DEFINE STORAGE command
Modify initial storage settings	DIRMAINT command[a]
Display storage allocation	QUERY VIRTUAL command

a. The Directory Maintenance Facility (DirMaint™) must be enabled in order to use DIRMAINT commands.

2.4.3 VDISKs

VDISKs are virtual disks, emulated disks that z/VM creates in virtual memory. Because they exist in main memory, VDISKs have very fast access times. Typical VDISK usage for Linux guests is a fast swap device (see Chapter 5, “Examining Linux swap device options” on page 47).

Important: We do not recommend using VDISKs for a Linux swap device if z/VM memory is constrained. The reason is that the page and segment tables that define the VDISK address space are not pageable. These tables take up 8 KB per 1 MB of VDISK size. Instead, put the Linux swap device on a minidisk that has a cached controller.

2.5 Influencing z/VM memory management

Several factors can influence how z/VM manages memory management. These include:

- **Ensure paging DASD, and the paths to them are not very busy.**
  This makes paging as fast as possible. Use multiple devices to permit overlapped I/O. Use dedicated full packs for paging DASD, and be sure to define enough so that you can do block paging.
- **Use expanded storage for paging.**
  As discussed 2.2, “Guidelines for allocation of z/VM storage” on page 12, expanded storage can improve overall performance.

- **Use shared segments.**
  Typically, Conversational Monitor System (CMS) is run from a named saved system (NSS), but other operating systems can be too. The advantage is that only one copy of the operating system resides in storage accessible to all virtual machines. We discuss creation of a Linux NSS in 4.2, “Exploiting the shared kernel” on page 37.
- **Adjust SRM controls.**
  These control how z/VM manages virtual memory. System Resource Management (SRM) parameters are discussed in 6.4.1, “Global SRM controls” on page 89.
- **Ensure system operation information records are properly retrieved and stored.**
  z/VM allows for collection of various accounting, error, and symptom records during operation. This data is stored in main memory until retrieved by designated virtual machines. A new installation of z/VM typically defines virtual machines that are automatically logged on at IPL:
  - The DISKACNT virtual machine retrieves accounting records.
  - The EREP virtual machine retrieves error records.
  - The OPERSYMP virtual machine retrieves symptom records.

  If these virtual machines stop retrieving the data, available system memory can greatly reduced. Further details on these information collection facilities can be found in *z/VM V4R3.0 System Operation*, SC24-6000.
- **Use the CP SET RESERVED command to reserve pages for a guest.**
  The CP SET RESERVED command reserves real storage pages to a virtual machine.
- **Using V=F or V=R.**
  These VM configurations allow main memory to be dedicated to a guest operating system; guest virtual machine memory pages reside permanently in main memory. The V=F option is only available if the hardware processor is IMLed in “basic” ESA/390 mode and not in LPAR mode.

  Important: There are limitations and restrictions associated with V=R and V=F. Both can adversely affect overall system performance.

Although these factors will influence virtual memory management, not all can be considered beneficial. In general, options that enable resource sharing are recommended for tuning overall system performance.

Note: While dedicated resources can be useful in some specific system designs, their use should be carefully considered before deployment. Dedicated virtual memory management options include:

- Using the CP SET RESERVED command.
- Using V=F or V=R.

This redbook concentrates on methods of resource sharing. Dedicated resources cannot be recommended.

2.6 Paging and spooling

z/VM utilizes specially allocated DASD space for system storage that is available to all virtual machines running within that system. This DASD storage is owned and controlled by the system (CP) and has distinct uses.

As already discussed, paging DASD storage is utilized for virtual machine paging along with main memory and expanded storage. Paging needs to be as fast as possible; distributing paging DASD space across numerous devices and channels is suggested. Because the data in page storage is temporary, all the paging data is discarded when z/VM is shut down.

Spool space is used to store data that is common to all virtual machines. Because z/VM simulates an environment of independent operating systems, each of these virtual machines has a designated reader, punch, and printer, just like the computer systems of old. Data sent to a virtual machine will reside in the reader until deleted or “read” into a user’s minidisk, much like a mail in-box. Data that is created as output is directed to the virtual printer or to the virtual punch.

Spool space is also used for executable data or systems stored for access of all virtual machines, such as NSS files. This creates a common location for system code, rather than each virtual machine requiring individual storage space. Because the contents of spool space is valid data, it is preserved across z/VM shutdowns and IPLs.

Along with DASD storage that is owned by specific virtual machines (better known as minidisks), z/VM allows for temporary minidisk space. This space can be allocated as needed by a virtual machine from the tdisk storage created on the system. This is meant to only hold data that is being processed or does not need to be is retained. When z/VM is shut down, all data in tdisk storage is discarded.

3

Linux virtual memory concepts

In this chapter, we discuss how Linux uses virtual memory. We present Linux memory management concepts and consider how these concepts affect Linux guests running under z/VM. Topics include:

- Components of the Linux memory model
- Linux memory management
- Observing Linux memory usage
- Illustrating Linux aggressive caching
- Conclusions for sizing z/VM Linux guests

3.1 Components of the Linux memory model

We begin examining Linux memory by looking at the components of memory allocation.

3.1.1 Linux memory

Linux manages its memory without regard to the fact that it is running in a z/VM virtual machine. The Linux kernel uses an “always full” concept of memory management. It attempts to load as much information (applications, kernel, cache) into its perceived memory resources as possible.

When Linux boots, one of its first tasks is the division and allocation of memory resources. Memory is divided into three main components:

- **Kernel memory**
  Kernel memory is where the actual kernel code is loaded, and where memory is allocated for kernel-level operations. Kernel operations include:
  - Scheduling
  - Process management
  - Signaling
  - Device I/O (including both to disk and to network devices)
  - Paging and swapping
- **User memory**
  User memory is where all application code is loaded.
- **Buffer and cache memory**
  The rest of the memory is used for caching both I/O and file system data. In the Linux 2.4 kernel, two types of caches are used:
  - Buffer cache
    The buffer cache contains the buffers Linux uses during handling of I/O requests.
  - File system cache
    The file system cache contains data from the files in the Linux file system (including the actual content of the files themselves). Typically, the bulk of Linux caching is done in the file system cache.

  Buffer and file system cache are intended to speed overall system performance by reducing I/O operations (which are inherently slower than memory access).

Note: Linux uses an aggressive caching policy intended to reduce I/O when allocating main memory; the theory being that memory is better utilized as cache, rather than leaving it unused and available. This policy is illustrated in 3.4, “Illustrating Linux aggressive caching” on page 30.

3.1.2 Linux swap space

Linux typically asks for the creation of a swap device during initial installation of a system. For periods of high memory demand, Linux will temporarily move less frequently accessed memory pages to its swap devices. The swapped memory pages then become available for use. When a memory page residing on a swap device is accessed, Linux moves it back into an available real memory page. Options for defining swap space for Linux on zSeries is discussed in Chapter 5, “Examining Linux swap device options” on page 47.

Note: Do not enable MDC on Linux swap minidisks. The read ratio is not high enough to overcome the write path length penalty.

Swapping versus paging

Swapping is the process of moving an entire address space to a swap device. *Paging* is the process of moving pages of memory to a swap device. In the past, a swap device was used for swapping. Because of its inherent inefficiency (swapping requires frequent, expensive context switches), swapping has been replaced with paging. Although Linux utilizes a paging algorithm, the name swap device has been retained.

3.2 Linux memory management

Linux memory is divided into *pages*, and each page is 4096 bytes in size. It is the responsibility of the Linux memory manager to control usage of these pages. A counter is maintained for each page and is used to determine whether to keep the page in real memory.

Portions of memory are scanned periodically to check current page usage:

- When the virtual memory scanner locates pages that can be removed from main memory, a counter on those pages is decreased by dividing the counter value by 2 (an exponential decline).
- When the scanner locates pages that should not be removed from memory (because the page was recently accessed), the page counter is increased by a constant value.

Eventually, less recently accessed pages acquire a lower counter value, and more recently accessed pages acquire a higher counter value. The counter value is used by the memory manager during page cleaning.

3.2.1 Page cleaning

Linux memory pages are cleaned up (swapped out) when:

- **The kswapd kernel thread runs.**
  This thread wakes up once a second to check the number of free page frames. If the number is below a threshold, inactive pages are moved to a swap device.

  In some Linux kernels, the kswapd thread also wakes up every five minutes to clean inactive pages from the buffer cache and inode cache.

- **A process requests more memory than is currently available.**
  The kswapd thread runs if a memory request cannot be fulfilled from the available memory pool. Less recently accessed pages are moved to the swap device in order to free the required memory.

  Thrashing occurs when not enough inactive pages are available to satisfy a memory request. In this case, Linux spends more time moving pages to and from the swapping device than running user processes.

Figure 3-1 on page 25 illustrates the effect of the Linux page cleaner.

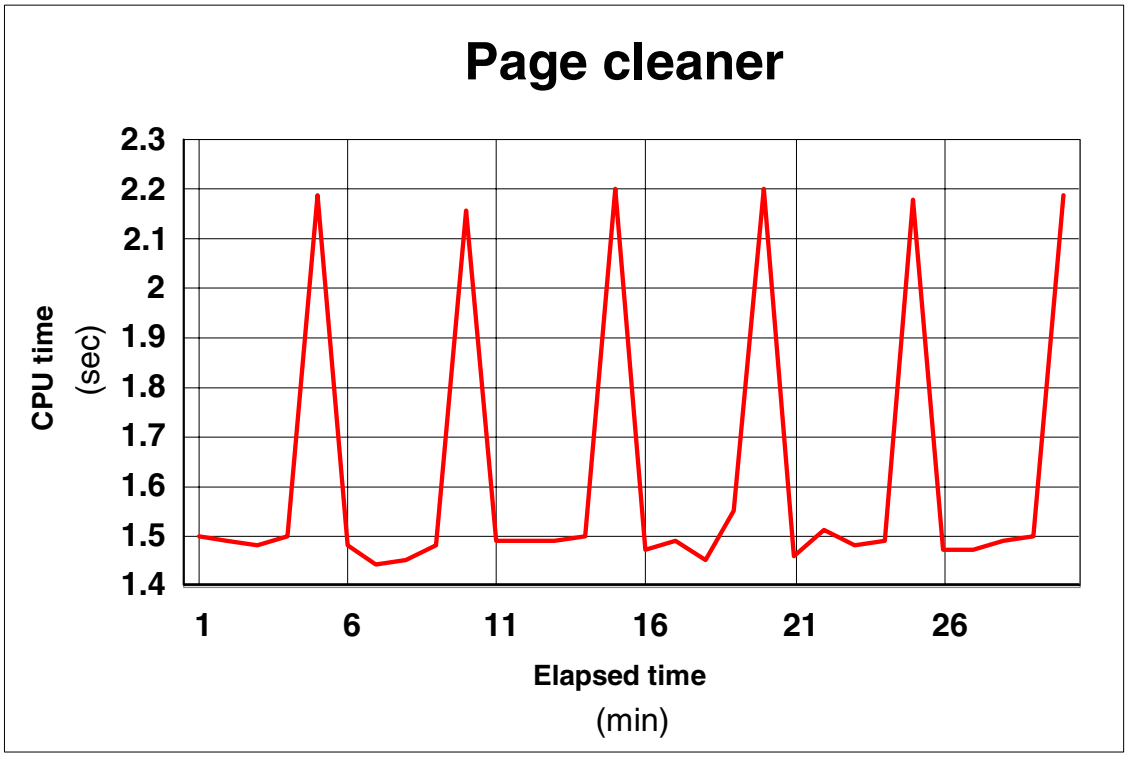


Figure 3-1 Illustrating the effect of page cleaning

In Figure 3-1, we see CPU utilization for an idle Linux guest plotted over time. The spike in CPU utilization at five minute intervals is due to the kswapd kernel thread performing its periodic memory cleanup.

3.3 Observing Linux memory usage

For a quick look at how Linux allocates memory, we use the Linux **`free -k`** command (the **`-k`** option reports memory size in kilobytes). Example 3-1 on page 26 illustrates the memory usage of a 128 MB virtual memory Linux guest.

Example 3-1 Observing Linux memory usage using the free command

```
$ free -k
             total       used       free     shared    buffers     cached
Mem:        121276      50768      70508          0       4832      26944
-/+ buffers/cache:      18992     102284
Swap:            0          0          0
```

Important points to note in Example 3-1:

- The total memory (121276 kB) is less than the total virtual memory size allocated to the Linux guest (128000 kB). The difference (6724 kB) is the size allocated to the kernel.
- The total memory (121276 kB) is equal to used memory (50768 kB) plus free memory (70508 kB).
- The used memory (50768 kB) is equal to buffer (4832 kB) plus cached memory (26944 kB) plus used buffers/cache memory (18992 kB).
- The used buffers/cache memory (18992 kB) plus free buffers/cache memory (102284 kB) is equal to total memory (121276 kB).
- The free buffers/cache memory (102284 kB) is equal to free memory (70508 kB) plus buffer memory (4832 kB) plus cache memory (26944 kB).

Although there is 102284 kB “free” memory available, Linux expects applications to use only 70% (70508 / 102284). Linux expects to use the remainder as buffer/cache memory.

3.3.1 Kernel memory usage at system boot

To examine kernel memory usage at system boot, we use the **demsg** command, as illustrated in Example 3-2.

Example 3-2 Memory usage by the kernel at system boot

```
$ demsg | less
.
.
Memory: 120188k/131072k available (1719k kernel code, 0k reserved, 843k data, 64k init)
Dentry-cache hash table entries: 16384 (order: 5, 131072 bytes)
Inode-cache hash table entries: 8192 (order: 4, 65536 bytes)
Mount-cache hash table entries: 2048 (order: 2, 16384 bytes)
Buffer-cache hash table entries: 8192 (order: 3, 32768 bytes)
Page-cache hash table entries: 32768 (order: 6, 262144 bytes)
.
.
```

In this example, we see Linux allocates inode and buffer cache at system boot.

3.3.2 Detailed memory usage reported by /proc/meminfo

To examine memory in detail, we query the /proc/meminfo kernel driver, as shown in Example 3-3.

Example 3-3 Detailed analysis of Linux memory using /proc/meminfo

```
$ cat /proc/meminfo
        total:    used:    free:  shared: buffers:  cached:
Mem:  124186624 48365568 75821056        0  4104192 25628672
Swap:        0        0        0
MemTotal:       121276 kB
MemFree:         74044 kB
MemShared:           0 kB
Buffers:          4008 kB
Cached:          25028 kB
SwapCached:          0 kB
Active:          27600 kB
Inact_dirty:      1436 kB
Inact_clean:         0 kB
Inact_target:    32768 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:       121276 kB
LowFree:         74044 kB
SwapTotal:           0 kB
SwapFree:            0 kB
```

Explanations of the fields shown in Example 3-3 are as follows:

- **MemTotal**
  The amount of RAM memory assigned to Linux, not including memory used by the kernel. In Example 3-3, Linux runs in a 128 MB virtual machine; however, only 121276 KB is available for user memory, buffers, and cache.
- **MemFree**
  Reports the current amount of memory not in use by Linux.
- **MemShared**
  This number is always zero in 2.4 kernels, because it is too expensive to actually calculate it. If calculated, it would report the sum of memory shared between processes.
- **Buffers**
  Reports the size of memory allocated to I/O buffers. Buffers hold data accessed from block devices.

- **Cached**
  Reports the size of memory allocated to cache. Caches hold data accessed from files.
- **SwapCached**
  Reports the size of cache memory swapped out to swap devices.
- **Active**
  Reports the size of active memory pages (pages that are frequently accessed).
- **Inact_dirty**
  Reports the size of dirty inactive memory pages:
  - Inactive pages are less frequently accessed relative to active pages (and therefore are eligible for swapping should a memory shortage arise).
  - Dirty pages are out of sync with respect to their backing store.

  When memory is required, Linux chooses to steal Inact_clean pages *before* swapping Inact_dirty pages.
- **Inact_clean**
  Reports the size of clean inactive memory pages. Because clean pages are in sync with respect to their backing store, Linux can reuse (steal) Inact_dirty pages without having to write the page to a swap device.
- **Inact_target**
  From reviewing the kernel source code, this number is described as the "number of inactive pages we ought to have." It is calculated as the sum of Active, Inact_dirty, and Inact_clean divided by 5. This is most likely an indication of when page cleaning should be performed.
- **HighTotal**
  This value reports the amount of MemTotal not directly mapped into kernel space. Its value varies based on kernel level. On zSeries, this value is always zero because the kernel is loaded into its own distinct space (thus allowing a 31-bit address space for user memory). Documentation can be found in the Debugging390.txt file in the kernel source documentation.
- **HighFree**
  Reports the amount of MemFree not directly mapped into kernel space. The value varies based on the kernel level. Its value is zero on zSeries.
- **LowTotal**
  Reports the amount of memory that is directly mapped into kernel space. The value varies based on the type of kernel used.
- **LowFree**
  Reports the amount of free memory that is directly mapped into kernel space. The value varies based on the type of kernel used.

- **SwapTotal**
  Reports the amount of swap space available.
- **SwapFree**
  Reports the amount of swap space not yet used.

Details about Linux Virtual Memory Management can be found at:

http://cs.uml.edu/~cgould
http://www.csn.ul.ie/~mel/projects/vm/guide/html/understand/

3.3.3 Using the vmstat command

The Linux **vmstat** command reports statistics about processes, memory, paging, I/O to block devices, and CPU usage. The command syntax is:

```
vmstat [delay [count]]
```

Where:

delay The delay (in seconds) between updates
count The number of updates

Example 3-4 illustrates the **vmstat** command output.

Example 3-4 Using the vmstat command

```
# vmstat 60 5
   procs                  memory      swap          io     system         cpu
 r  b  w   swpd   free   buff  cache   si   so    bi    bo   in    cs  us  sy  id
18  0  6  27260   1020     60    724 3889 4981  4057  4981    0   653  82  18   0
13  0 11  27228   2036     64    772 3802 4868  3970  4868    0   633  82  18   0
16  0  8  27228   1200     64    740 3713 4802  3868  4802    0   621  82  18   0
17  0  7  27228   1332     64    888 3719 4703  3874  4703    0   676  82  18   0
14  0 10  27236   1796     60    724 3741 4798  3900  4798    0   633  82  18   0
#
```

Statistics reported by the **vmstat** command are grouped by type:

- **procs**
  Process statistics are reported as the average number of processes in state:
  r Number of processes waiting for run time
  b Number of processes in uninterruptable sleep state
  w Number of processes swapped out but otherwise runnable
- **memory**
  Memory statistics are reported as the average amount of memory:
  swpd Used memory size (KB)

free Unused memory size (KB)

buff Memory allocated to buffer cache (KB)

cache Memory allocated to file system cache (KB)

- **swap**
  Paging statistics report average paging rates to swap devices:

  si Paging rate from swap device to memory (KB/s)

  so Paging rate from memory to swap device (KB/s)

- **io**
  I/O statistics report the average I/O rate to block devices:

  bi Number of blocks sent to a block device (blocks/s)

  bo Number of blocks received from a block device (blocks/s)

- **system**
  System statistics report average system activity:

  in Number of interrupts per second (including clock interrupts)

  cs Number of context switches per second

- **cpu**
  CPU statistics report average utilization (as a percentage) of the CPU:

  us Time spent in user mode

  sy Time spent in kernel mode

  id Time spent idle

Note: Be aware that when running as a z/VM guest, the numbers reported by Linux utilities, such as **`vmstat`**, assume the guest owns 100% of the system resources. In reality, these resources are shared by all virtual machines running in the z/VM image. Linux resource counters report values relative to the virtual machine in which they operate.

3.4 Illustrating Linux aggressive caching

To illustrate Linux aggressive caching, we compare the memory usage of two Linux guests, one running in a 64 MB virtual machine, the other in a 128 MB virtual machine. Each runs the Mstone workload over an eight minute interval (the Mstone workload is discussed in Appendix B, “Mstone workload generator” on page 159). In Figure 3-2 on page 31, we examine memory usage over time.

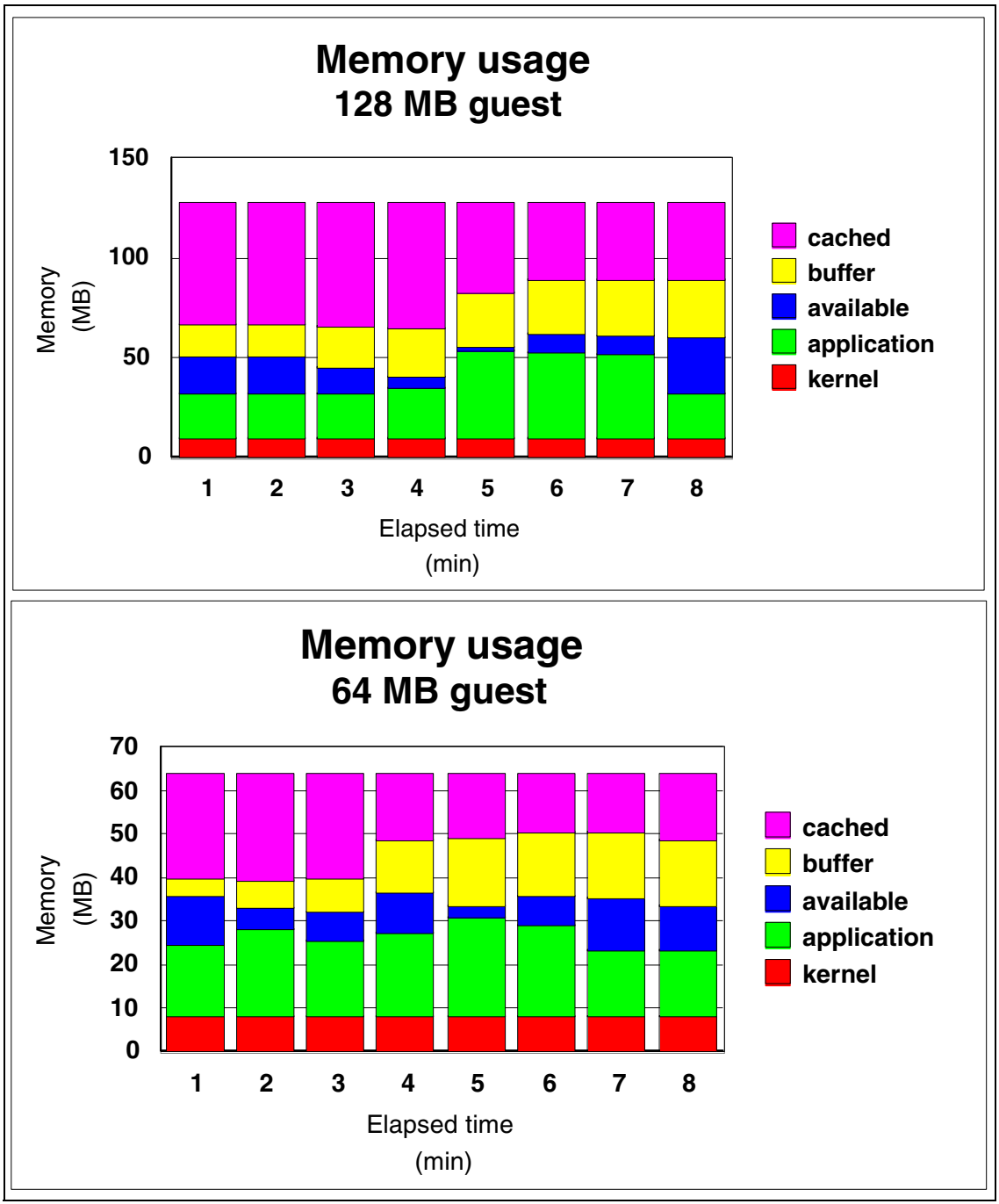


Figure 3-2 Linux memory usage in two Linux guests: 128 MB and 64 MB

Comparing the two Linux guests, we see a similar memory usage pattern: In both cases, additional application memory is obtained at the expense of buffer and cache memory. Reducing the virtual machine size by 50% reduced average caching by 60%.

Note: Although the 64 MB guest required half the amount of memory, no appreciable effect on server response time was noted.

3.4.1 Choosing the correct virtual machine size

Figure 3-3 illustrates the effect of virtual memory size on performance.

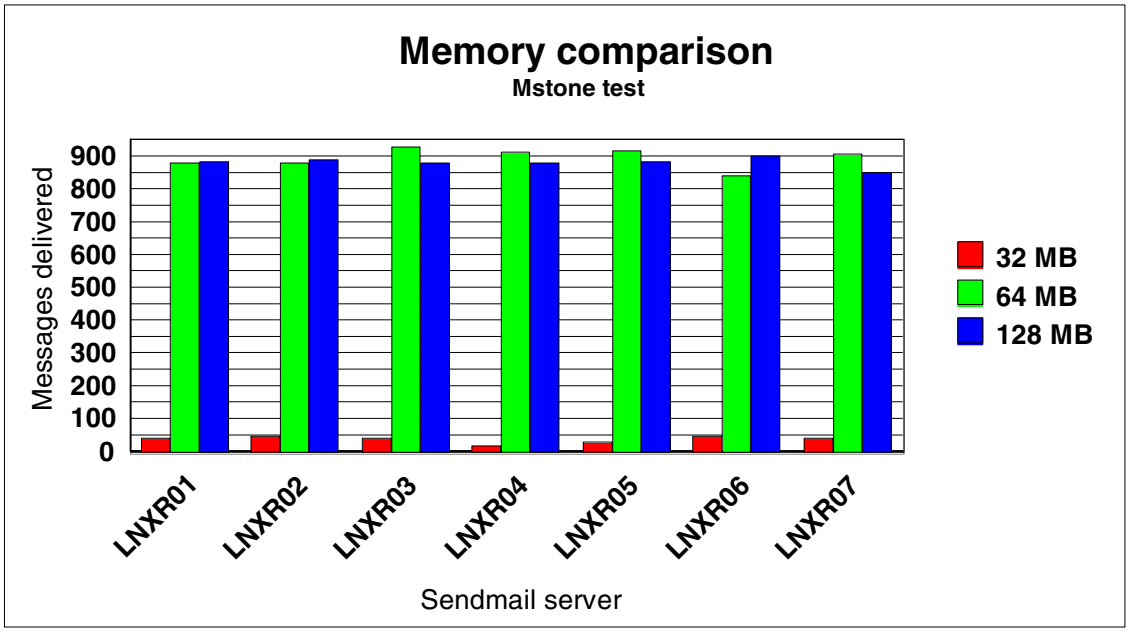


Figure 3-3 Mstone performance for 32 MB, 64 MB, and 128 MB Linux guests

The Mstone benchmark is run against Linux guests with virtual memory sizes of 32 MB, 64 MB, and 128 MB. The total number of delivered messages is used to gauge how well sendmail operates in the available Linux memory. From Figure 3-3, we see there is no significant difference in running sendmail in a 64 MB or 128 MB virtual machine. However, in a 32 MB virtual machine, the sendmail server becomes overloaded.

Important: Even though the 128 MB server does not require all that memory, it will eventually appear to use it all. Its memory cost is four times that of the 32 MB server.

3.5 Conclusions for sizing z/VM Linux guests

The Linux memory model has profound implications for Linux guests running under z/VM:

- **z/VM memory is a shared resource.**
  Although aggressive caching reduces the likelihood of disk I/O in favor of memory access, the cost of caching must be considered: Cached pages in a Linux guest *reduce* the number of z/VM pages available to other z/VM guests.
- **A large virtual memory address space requires more Linux kernel memory.**
  A larger virtual memory address space requires more kernel memory for Linux memory management.

When sizing the memory requirements for a Linux guest, choose the smallest memory footprint that has a minimal effect on performance.

Tip: To determine the smallest memory footprint required, decrease the size of the Linux virtual machine to the point where swapping begins to occur under normal load conditions. At that point, slightly increase the virtual machine size to account for some additional load.

To reduce the penalty of occasional swapping that might occur in a smaller virtual machine, use fast swap devices, as discussed in Chapter 5, “Examining Linux swap device options” on page 47.

4

Tuning memory for z/VM Linux guests

In this chapter, we discuss tuning memory and storage for z/VM Linux guests. We make memory tuning recommendations and show how to use a shared Linux kernel to reduce memory usage. Topics include:

- Memory tuning recommendations
- Exploiting the shared kernel

4.1 Memory tuning recommendations

Because storage is limited, every option should be taken to reduce storage requirements. To illustrate, if 100 servers each require 100 MB of memory (a conservative estimate), 10 GB of real storage would be required. For 500 servers, the requirement would be 50 GB, pushing the limits of what is currently available.

4.1.1 Reduce storage of idle servers

The cost of idle servers should be minimized. If the cost of an idle server is 100 MB of real storage, any option to reduce this is important.

Apply the timer patch to Linux guests. The default Linux scheduler wakes up 100 times per second. This does not allow idle servers to drop from queue, and VM will not be able to trim the working sets of the idle servers. Install the timer patch to reduce the storage of idle servers. The effect of the timer patch is examined in 7.3, ”The Linux timer patch” on page 108.

Note: For z/VM V4.3 and earlier, Linux machines with queued direct input/output (QDIO) and channel-to-channel (CTC) devices (virtual as well as dedicated) do not drop from queue even with the timer patch applied. The PTF for APAR VM63282 is expected to resolve this problem. Until this PTF is applied, use of Inter-User Communications Vehicle (IUCV) connections is recommended for idle Linux guests running in a constrained environment. Be aware that this PTF is not a substitute for the timer patch: Without the patch, the PTF would be ineffective.

4.1.2 Reduce operational machine sizes

There are several opportunities for reducing the size of operational storage:

- **Eliminate unneeded processes.**
  This might sound intuitive, but it needs to be done. Processes such as **`cron`** should be eliminated when they do not perform useful functions. The impact of running unneeded processes is demonstrated in 7.2, ”The effect of idle servers on performance” on page 105.
- **Divide large servers into smaller specialized servers.**
  Service machines can be tuned explicitly to perform a function. Linux servers running many applications do not have these controls and can be difficult to manage. Separating functions into smaller Linux guests can be done at little cost; this can make the servers easier to tailor.

- **Reduce machine size.**
  Minimize the virtual machine size to the lowest possible amount. Choosing this amount requires research and testing. Because Linux will use all available storage, defining a smaller machine size reduces real storage requirements.
- **Use a virtual disk for swap device.**
  Using a virtual disk as a swap device reduces the performance penalty of swapping.

4.1.3 Reduce infrastructure storage costs

Several opportunities for reducing infrastructure storage costs exist:

- **Use DIAGNOSE driver for DASD and MDC record cache.**
  Using a record-level minidisk cache will reduce the amount of storage required for MDC. This requires the DIAGNOSE driver.
- **Use shared storage.**
  Opportunities to share storage between many servers reduces real storage requirements by the amount of storage shared by each server. In 4.2, ”Exploiting the shared kernel” on page 37, we show how to create a Named Saved System (NSS) for Linux.

4.2 Exploiting the shared kernel

We believe it is important for the scalability of Linux on z/VM to share the Linux kernel in NSS. However, although the ability to share the Linux kernel is part of the official S/390 Linux source tree, a compiled kernel that exploits this feature is not provided in the SuSE or Red Hat distributions.

Note: To create a Linux system that utilizes NSS support, you need to recompile the Linux kernel. Be aware that your distributor might not support systems running with a custom compiled kernel. Because this might be the case, NSS support should be considered experimental. Depending on the level of support required, this approach might not be appropriate for your installation.

To create a Linux NSS:

1. Compile the Linux kernel with the appropriate configuration options for NSS enabled.
2. Define a skeletal system data file (SDF) for the compiled kernel using the CP DEFSYS command.

3. Save the NSS-enabled kernel into the SDF using the CP SAVESYS command.

The kernel configuration option CONFIG_SHARED_KERNEL is used to enable the kernel code and data in memory segments. This is necessary because the shared kernel code must be protected against updates from the Linux images that use that kernel. In S/390 architecture, protection is assigned on a segment basis (each segment is 1 MB in size).

Note: When Linux is running in dedicated memory (for example, on an Intel server or in a S/390 LPAR), this option would not be used; it would only waste some of the memory dedicated for use by that server. However, when running Linux in a virtual machine, there is very little cost involved with virtual memory that is never used. It does not have to be provided by z/VM either (the lost memory below 3 MB can be compensated for by giving the virtual machine slightly more memory).

As shown in the Linux memory map in Figure 4-1 on page 39, there is some unused address space between the R/O and R/W portions of memory.

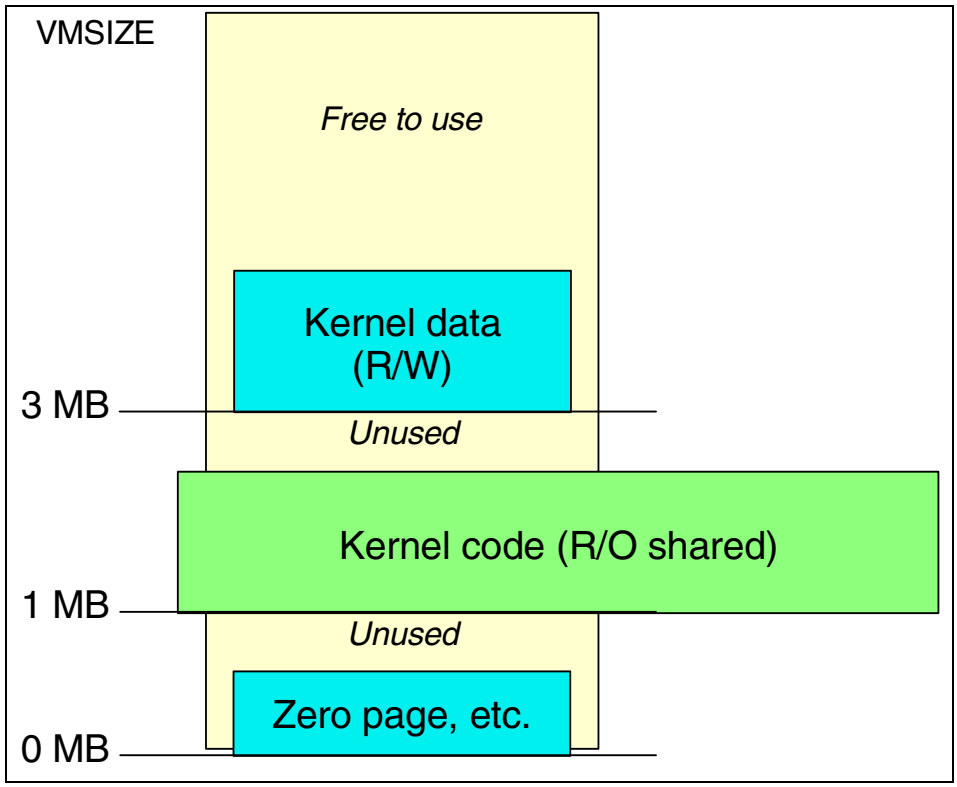


Figure 4-1 Linux memory map with shared kernel

4.2.1 Building an NSS-enabled Linux kernel

To build a Linux kernel for zSeries or S/390:

1. Obtain the kernel source code.
2. Obtain and apply the zSeries- and S/390-specific patches and Object Code Only (OCO) modules.
3. Configure and build the kernel.
4. Copy the OCO modules to the appropriate locations in the file system.
5. Copy the new kernel to the /boot directory and run the **`zipl`** command.

Details about building a Linux kernel for zSeries can be found in *Linux on IBM @server zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP3596 at:

`http://www.ibm.com/redbooks/abstracts/redp3596.html`

For NSS support, we need to enable the CONFIG_SHARED_KERNEL option. Using the **make menuconfig** command, this option can be enabled by selecting **General Setup** → **VM shared kernel support**. Example 4-1 shows the main screen with General Setup option.

Example 4-1 Main screen: make menuconfig

```
Code maturity level options  --->
Loadable module support  --->
Processor type and features  --->
General setup  --->
SCSI support  --->
Block device drivers  --->
Multi-device support (RAID and LVM)  --->
Character device drivers  --->
Network device drivers  --->
Miscellaneous  --->
Networking options  --->
File systems  --->
Kernel hacking  --->
---
Load an Alternate Configuration File
Save Configuration to an Alternate File
```

In the General setup menu shown in Example 4-2 on page 41, select the **VM shared kernel support** option.

Example 4-2 General setup: VM shared kernel support

```
[*] Fast IRQ handling
[*] Process warning machine checks
[*] Use chscs for Common I/O
<M> QDIO support
[ ]    Performance statistics in /proc
[*] Builtin IPL record support
(vm_reader) IPL method generated into head.S
[*] Networking support
[*] System V IPC
[ ] BSD Process Accounting
[*] Sysctl support
<*> Kernel support for ELF binaries
< > Kernel support for MISC binaries
[ ] Show crashed user process info
[*] Pseudo page fault support
[*] VM shared kernel support
[*] No HZ timer ticks in idle
[ ]   Idle HZ timer on by default
```

One should not underestimate the cost of compiling the kernel. On a zSeries CPU, a complete build of the kernel may well use some 10 to 15 minutes of CPU time. If you worry about the cost of an idle Linux virtual machine, a trimmed down Linux image can run idle for more than a month on the amount of CPU cycles the kernel build takes.

Important: At least with the Linux 2.4.7 kernel sources, you should run a **`make clean`** after you changed the `CONFIG_SHARED_VM` option. There appears to be a problem with the build process in that it does not pick up the change.

4.2.2 Defining a skeletal system data file for the Linux NSS

We examine the generated System.map file (shown in Figure 4-1 on page 39) to identify the zero page, shared kernel code, and non-shared kernel data regions in the constructed kernel.

Example 4-3 Portions of the System.map to determine NSS configuration

```
00000000 A _text                                    1
00000298 t iplstart
00000800 T start
00010000 t startup
00010400 T _pstart
00011000 T _pend                                    2
00100000 T _stext                                   3
001000a8 t rest_init
001000fc t do_linuxrc

002d5918 A __stop___kallsyms
002d5918 A __stop___ksymtab                         4
00300000 A _etext                                   5
00300000 d init_mmap
00300040 d init_fs
00300064 d init_files

00381700 d sockets_in_use
00381800 D softnet_data
00381900 D tcp_hashinfo
00382000 A __bss_start                              6
00382000 b totalram_pages
00382004 b pseudo_wait_spinlock
00382008 b ext_int_pfault
```

We find the page range for each region based on the symbols that indicate the start and end of the region.

Note: The address we use to locate the end of region is actually one byte past the end of the region. To compensate, we subtract one byte in the calculation. Because pages are 4 K in size, we calculate the page range by dropping the last three nibbles from the address.

From the System.map file, we identify:

- The region extending from `_text` (1) to `_pend` (2) is the zero page area. Using addresses 0x00000000 and 0x00010fff, we find the page range is 0-10.
- The region extending from `_stext` (3) to the symbol *immediately before* `_etext` (`__stop__ksymtab` 4) is the shared kernel code. Using addresses 0x00100000 and 0x002d5917, we find the page range is 100-2D5.
- The region extending from `_etext` (5) to `_bss_start` (6) is the non-shared kernel data. Using addresses 0x00300000 and 0x00381fff, we find the page range is 300-381.

From these values, we construct the following DEFSYS command:

```
DEFSYS SUSE72 0-10 EW 100-2D5 SR 300-381 EW MINSIZE=24M MACHMODE XA,ESA
```

Parameters to the command are:

- **SUSE72**
  The name to identify the NSS on IPL. By using different NSS names, you can have different versions of the kernel saved as NSS on a VM system. The IPL command identifies the NSS to be used by the virtual machine. Use this ability with care. Sharing is most efficient (both in memory and system administration) when many virtual machines share the same NSS.
- **0-10 EW**
  The page range for the zero page region. This region is copied to each virtual machine in “exclusive write” mode (EW).
- **100-2D5 SR**
  The page range for the shared kernel code. Each virtual machine uses a shared, read-only copy (SR).
- **300-381 EW**
  The page range for the private kernel data. Each virtual machine uses a “exclusive write” copy.
- **MINSIZE=24M**
  The minimum virtual machine size to use the NSS (only to prevent an IPL in a virtual machine where it will not fit anyway).
- **MACHMODE XA,ESA**
  The NSS can be IPLed in an XA or Enterprise Systems Architecture (ESA) virtual machine.

4.2.3 Saving the kernel in the Linux NSS

The NSS-enabled kernel is saved into the skeletal SDF using the SAVESYS command. As an example, we use the SAVELX EXEC script from the “How To Use VM Shared Kernel” Support page at:

```
http://www.vm.ibm.com/linux/linuxnss.html
```

We modify the script to use the DEFSYS command parameters specific to our kernel, as shown in Example 4-4 on page 44.

Note: The original values used for the DEFSYS command will work when creating the SDF. However, these values create a larger NSS than required (and therefore require more time to IPL).

Example 4-4 SAVELX EXEC to save a Linux NSS

```
/* SAVELX EXEC */
/* get the system name and device to ipl */
parse arg lxname devnum
lxname = strip(lxname)
devnum = strip(devnum)
/* figure out the line end character */
'pipe cp q term | var termout'
parse var termout one myend three
myend = strip(myend)
/* figure out the storage size */ 'pipe cp q v stor | var storout'
parse var storout one two storsize
/* construct the defsys command */
DODEF = 'DEFSYS' lxname '0-10 EW 100-2D5 SR 300-381 EW MACHMODE XA,ESA'
dodef = dodef 'MINSIZE=' || storsize
say dodef
/* define the saved system */
dodef
/* arrange to stop the ipl processing at the appropriate spot, */
/* at which point a savesys will be issued */
SETSAVE = 'TRACE I R 010000 CMD SAVESYS' lxname
setsave = setsave || myend 'TRACE END ALL'
say setsave
setsave
doipl = 'i' devnum
say doipl
/* all set, issue the ipl */
doipl
exit
```

SAVELX EXEC requires two input parameters:

1. The name of the Linux NSS. In our example, we use the name SUSE72.
2. The DASD device on which the NSS-enabled kernel resides. This DASD device should be defined for each virtual machine using the Linux NSS.

Assuming the NSS-enabled kernel resides on virtual device 201, we save Linux using:

```
SAVELX SUSE72 201
```

4.2.4 Changing Linux images to use the shared kernel in NSS

To boot the Linux NSS in a virtual machine, issue the CP IPL command using the name of the Linux NSS. In our example, the NSS is named SUSE72:

```
IPL SUSE72
```

Be aware that although the NSS-enabled kernel is not booted from DASD, DASD device numbers for virtual machines using the NSS must match the device numbers used when the NSS was created. For example, if the NSS-enabled kernel was created using a 201 disk as the root file system device, virtual machines using the NSS must define a 201 disk with a root Linux file system.

5

Examining Linux swap device options

In this chapter, we examine options for setting up a Linux swap file. Topics include:

- Linux swapping
- Swapping with ECKD discipline
- Impact of page-cluster on MDC hit rate
- The FBA discipline
- The DIAGNOSE discipline
- Using DIAGNOSE I/O for VDISK
- Using multiple VDISKs for swapping

In 5.7, “Linux swap device recommendations” on page 72, we make some general recommendations for creating a Linux swap device. We note the DASD storage devices used are RAMAC Virtual Array (RVA) units, not newer generation IBM TotalStorage® Enterprise Storage Server® (ESS) units.

5.1 Linux swapping

Over time, a Linux system will use all the available memory. When it does not need the memory to run processes, it will use all excess memory to cache data from disk. The most obvious way to prevent this is not to give the virtual machine more memory than it needs. Unfortunately, this is not always possible; memory requirements vary over time as processes start and stop. Consequently, the virtual machine will be either too big or too small.

When the virtual machine is too small for the workload, Linux will start to swap. Pages from other processes are moved out and are stored on the Linux swap disks to make room for the pages of the process that need to run. If Linux swaps continuously, performance will be affected (this is what in z/VM would be called paging). Occasional swapping is not necessarily bad; the acceptable amount of swapping depends on the efficiency of the swapping mechanism.

To compare the efficiency of swap device types for Linux, we ran a number of hogmem processes in a 32 MB virtual machine. The hogmem program is very simple; the listing can be found in 5.8, “Program text for hogmem” on page 73. It allocates the specified amount of virtual memory for the process and then sequentially accesses each page. When the amount of memory allocated by hogmem exceeds the free memory in Linux, some other things will be swapped out by Linux. When we run enough hogmem programs in parallel and allocate more virtual memory than what Linux can free up for us, constant swapping will occur (which is bad in real life).

With three different driver disciplines in Linux, and two different types of disks, we have four out of six combinations to try, as illustrated in Table 5-1.

Table 5-1 Driver disciplines for Linux swap devices

	3390	**VDISK**
ECKD™	Default for 3390	N/A
FBA	N/A	Default for VDISK
DIAGNOSE	MDC benefit	Efficient

The DIAGNOSE discipline of the Linux driver uses a high-level block I/O protocol (DIAG 250) to have CP perform the actual I/O operations when necessary. The measurements will show the benefit of this protocol over the defaults chosen by the driver. Before we go into detail about the DIAGNOSE discipline of the driver, we first set the baseline for our benchmark by measuring the extended count key data (ECKD) and fixed block architecture (FBA) discipline.

When we IPL the kernel in a 32 MB virtual machine, the output of `free`, as shown in Example 5-1, suggests that more than half of that storage should be available for processes.

Example 5-1 Available memory in an idle 32 MB virtual machine

```
             total       used       free     shared    buffers     cached
Mem:         27748      15104      12644          0        576       7952
-/+ buffers/cache:       6576      21172
Swap:       143796          0     143796
```

The output shows that 12644 KB is not in use. Much of the buffers and cache could be reclaimed by Linux when necessary, so 21172 KB is close to what we can obtain without causing much swapping.

After starting a 20 MB process, we reexamine memory usage in Example 5-2.

Example 5-2 Available memory when running a single 20 MB process

```
             total       used       free     shared    buffers     cached
Mem:         27748      25712       2036          0         48       1480
-/+ buffers/cache:      24184       3564
Swap:       143796      17192     126604
```

This shows that Linux did not give up all buffers and cache but decided to move some things to swap instead:

- Linux obtained 10 MB from the free pool, leaving 2036 KB available to satisfy sudden and urgent memory requests.
- The remaining 10 MB was obtained from buffers/cache and by memory freed by swapping.

As shown in Example 5-3, **`vmstat`** reports very little swapping while the process runs.

Example 5-3 Monitoring swap activity

```
   procs                  memory    swap          io     system          cpu
 r  b  w   swpd   free   buff  cache  si  so    bi    bo   in   cs  us  sy  id
 1  0  0  23668   2036     28    828   2   2     5     2    0   11  99   1   0
 1  0  0  23668   2056     28    748  13   2    24     2    0   11  99   1   0
 1  0  0  23668   2052     36    748   0   0     1     0    0   10 100   0   0
 1  0  0  23668   2052     36    748   0   0     0     0    0   10  99   0   0
```

To cause some swapping, we must push a bit harder. If pushed hard enough, we can obtain a fairly constant swap rate that can be studied with the different measurement tools.

5.2 Swapping with ECKD discipline

The output in Example 5-4 shows what happens when we increase the virtual memory to 21 MB. Because our program walks through the allocated memory sequentially, we force all pages from swap into memory, and out to swap again.

Example 5-4 Pushing Linux to swap

```
   procs                  memory      swap          io     system         cpu
r  b  w   swpd   free   buff  cache   si   so    bi    bo   in    cs  us  sy  id
1  0  1  25464   1852     68   2304  482  686   528   687    0    66  40   1  59
0  1  1  25564   2036     32    900 1972 1813  2052  1814    0   208   5   2  93
0  1  0  25564   1912     48   1264 1844 1962  1993  1962    0   196   5   2  93
0  1  1  25564   1960    132   2116 1796 1998  2051  2004    0   196   4   1  94
0  1  0  25564   1840     48   1264 2042 2055  2171  2055    0   211   4   2  94
0  1  1  25564   1836     48   1264 1951 1898  2079  1898    0   194   4   2  94
0  1  0  25564   2000     36   1248 1780 2038  1907  2038    0   190   4   1  95
1  0  1  25564   2036     40   1388 2070 1955  2218  1955    0   202   4   1  95
0  1  0  25564   1828     48   1244 1750 2102  1879  2102    0   189   5   2  93
0  1  0  25564   1696     36   1116 1810 1946  1932  1946    0   194   4   1  94
```

In Example 5-4, we note:

- **Swap rate averages slightly less than 2 MB/s.**
  Swap rate is reported in columns `si` (memory swapped in from disk) and `so` (memory swapped out to disk). These values are reported in terms of the number of 1 KB blocks transferred.
- **CPU utilization drops significantly once swapping begins.**
  CPU idle time (as reported in the `id` column) increases to 95% from its previous value of 0%. This is not because the process suddenly became more efficient; the process is simply suspended after a page fault.

Using only 5% of the CPU cycles in the previous example, we conclude that only 5% as much work is accomplished: Throughput is *reduced* to 5% or less due to swapping.

Based on Linux measurements, we cannot determine if the 3 to 4 MB/s swap rate is high; instead, we use z/VM measurements to make that determination. Example 5-5 on page 51 shows the z/VM I/O rate attributed to the Linux guest.

Example 5-5 Effect of MDC on Linux swapping

```
Screen: ESAUSR3  ITSO                          ESAMON V3.3  02/17 17:20-17:28
1 of 2  User Resource Utilization - Part 2     USER RMHTUX02        2064 C0ECB

                        DASD MDisk Virt Cache I/O    <---Virtual Device---->
         UserID    DASD Block Cache Disk   Hit Prty   <----I/O Requests----->
Time     /Class     I/O   I/O  Hits  I/O   Pct Queued  Cons   U/R  CTCA Other
-------- -------- ----- ----- ----- ---- -----  ----- ----- ----- ----- -----
17:28:00 RMHTUX02  2884     0    73    0   2.5      0     0     0     0     0
17:27:00 RMHTUX02  2811     0    89    0   3.2      0     0     0     0     0
17:26:00 RMHTUX02  2825     0    66    0   2.3      0     0     0     0     0
17:25:00 RMHTUX02  2660     0   322    0  12.1      0     0     0     0     0
17:24:00 RMHTUX02  2646     0   495    0  18.7      0     0     0     0     0
17:23:00 RMHTUX02  3022     0   579    0  19.2      0     0     0     0     0
17:22:00 RMHTUX02  3040     0   537    0  17.7      0     0     0     0     0
17:21:00 RMHTUX02  3137     0   673    0  21.5      0     0     0     0     0
```

In Example 5-5, we note:

- **The 4 MB/s swap rate translates to approximately 50 I/Os per second.** The DASD I/O rate is the total number of reads and writes. Therefore, the 4 MB/s swap rate from Example 5-4 on page 50 translates to some 3000 I/Os per minute, or 50 I/Os per second (from the 17:22 to 17:23 interval).
- **Some 20% of swapping initially comes from the MDC.** Example 5-5 illustrates what happens when MDC is disabled for the swap device. We see the MDC hit ratio drop from 20% to just 2% (be aware that more recent intervals are shown toward the top of the listing). The 2% ratio can be attributed to the hits against the root file system that resides on a minidisk allocated on the same volume.

> **Tip:** It is normally not wise to allocate the swap device of the Linux virtual machine on the same volume that holds the data of that same virtual machine. If you allocate them on different volumes, you spread the I/O and can exploit multiple paths to the DASD controller.

It might be somewhat surprising that MDC is not more effective in this case, even though only 21 MB of data is involved. When allocating pages on the swap disk, Linux tries to minimize seek distances. The effect for this workload is that the area of active pages sweeps over the total swap device. So from a cache point-of-view, the entire 140 MB of swap space is involved.

However, the VM MDC is a write-through cache. For write operations:

- If a track or block that exists in the cache is updated, the cache copy is first updated. The underlying storage media is then written before the I/O operation is declared complete.
- If the data to be written is eligible for MDC, but does not currently exist in the cache, the data is written to the underlying storage media but is *not* added to the cache.

Data is only added the cache when it is read from disk, not written to disk. In this test, swapping does not reference the same page twice (a page is swapped in to update its content from disk, a new page is then swapped out). Any benefit derived from the MDC is due to the fact that the MDC is set for full track caching; an entire track is read from disk when even a single block is requested. For this case, swap pages that exist in the MDC were added when a page on the same track was swapped in.

Note: As discussed in 3.1.2, "Linux swap space" on page 23, we recommend disabling MDC for Linux swap devices.

Now that we have determined how much z/VM I/O can be attributed to the Linux guest, we next look at the device to determine how much more throughput is possible. Example 5-6 shows the I/O measurements for the logical volume containing the swap device.

Example 5-6 Linux guest swapping though ECKD discipline of the driver

```
Screen: ESADSD2A ITSO                            ESAMON V3.3  02/17 17:10-17:13
1 of 3  DASD Performance Analysis - Part 1       DEVICE 3752          2064 COECB


          Dev        Device %Dev <SSCH/sec-> <-----Response times (ms)--->
Time      No. Serial Type   Busy   avg  peak  Resp  Serv  Pend  Disc  Conn
-------- *--- ------ ------ ---- *---- ----- ----- ----- ----- ----- -----
17:13:00 3752 LNXU4R 3390-3 94.1  50.6  50.6  18.6  18.6   0.2   1.2  17.2
17:12:00 3752 LNXU4R 3390-3 95.9  51.0  51.0  18.8  18.8   0.2   1.3  17.4
17:11:00 3752 LNXU4R 3390-3 95.5  50.5  50.5  19.3  18.9   0.3   1.3  17.3
```

We see the same 50 I/O operations per second as noted in Example 5-5 on page 51; it appears there is little competition for the volume from other users. This I/O rate has driven the DASD to 95% utilization; there is little chance of driving it harder this way.

Note: For a proper correlation of the I/O reported for the user and the device, your favorite performance monitor might offer an option to do seek analysis. This analysis can show the device I/O by minidisk. In our case, we use a simplistic test with little activity by other users; this makes it easy to draw conclusions.

5.2.1 Effect of the number of processes on Linux swapping

To get a better understanding of the relationship between the I/O rate and MDC hit ratio observed in Example 5-5 on page 51, we design another experiment. Once again, we use the hogmem program to stress Linux memory management. For this experiment, we run three tests:

Test 1 x 21 MB One hogmem process, using 21 MB of memory

Test 3 x 7 MB Three hogmem processes, each using 7 MB of memory

Test 21 x 1 MB Twenty-one hogmem processes, each using 1 MB of memory

For each test, we use a 200-cylinder disk used with the ECKD driver as a swap device. Table 5-2 shows some of the disk I/O characteristics for our three swapping experiments.

Table 5-2 Disk I/O metrics for the three tests

	Test 1 x 21 MB	Test 3 x 7 MB	Test 21 x 1 MB
Linux swap-in (KB/s)	1960	1927	1868
Linux swap-out (KB/s)	1940	1892	1605
DASD I/O rate (SSCH/s)	52.8	76.0	196
MDC hit ratio (%)	22.3	23.7	38.6
Device utilization (%)	95.1	99.5	99.2
Device response time (ms)	18.6	14.7	7.6

One of the interesting differences is the reduced disk response time. Because the device utilization is close to 100%, there is a direct relation to the I/O rate as well. When we look at the Linux swap rate, there is no direct relation (if anything, the amount of data swapped by Linux decreases a bit where the I/O rate is almost four times higher). The first conclusion must be that the “size” of each DASD I/O in the Test 21 x 1 case apparently is less than of the Test 1 x 21.

Fortunately, VM provides options to study the I/O in more detail. During each test, we take a short I/O trace using the TRACE IO 200 CCW PRINTER command. This trace shows details for each I/O operation. The first part of one “channel program” generated by Linux is shown in Example 5-7.

Example 5-7 Output of the TRACE IO command

```
-> 0020BDC0' SSCH  B2333000    004C1694    CC 0    SCH 0004    DEV 0200
            CPA 00C68A90   PARM 004C1614    KEY 0  FPI C0  LPM F0
VDEV 0200 CCW 63400010 00C68A70
CCW    00C68A90  63400010 00C68A70   0000  63400010 ........
 EXTENT                 80CC0000 00000000 00220009 0022000A
CCW    00C68A98  47400010 00C68A80   0008  47400010 ........
 LOCATE RECORD          0180000C 00220009 00210009 043B1000
CCW    00C68AA0  85401000 00741000   0010  85441000 ........
 IDAL                                     780A1000
 IDAL                                     780A1800
CCW    00C68AA8  85401000 0115C000   0018  85441000 ........
 IDAL                                     429D2000
 IDAL                                     429D2800
CCW    00C68AB0  85401000 01D76000   0020  85441000 ........
 IDAL                                     342D2000
 IDAL                                     342D2800
```

Each channel program starts with a “define extent” and “locate record” channel command word (CCW). These two define the area on DASD where data is read or written. One CCW for each consecutive block read or written follows; the *more* blocks read or written in a single I/O, the *longer* the channel program gets.

For each test, we capture 50,000 records of trace information and use a set of *CMS Pipelines* filters to analyze the data. The easiest way to analyze the data is to compute average channel program length and the read/write ratio. These values are shown in Table 5-2 on page 53.

Table 5-3 Start subchannel (SSCH) count and read/write ratio

Test case	Channel programs			Blocks		
	Read	Write	Read/write	Read	Write	Read/write
Test 1 x 21 MB	562	119	4.7	7300	6666	1.1
Test 3 x 7 MB	708	267	2.6	6799	6595	1.0
Test 21 x 1 MB	1277	786	1.6	5933	5326	1.1

When we look at Table 5-3, we see that, for example, the length of write channel programs changes much more than the length of read channel programs. This means the average length does not help us a lot.

From the same VM I/O traces, we can obtain even more detailed statistics, as shown in the graphs in Figure 5-1 on page 55.

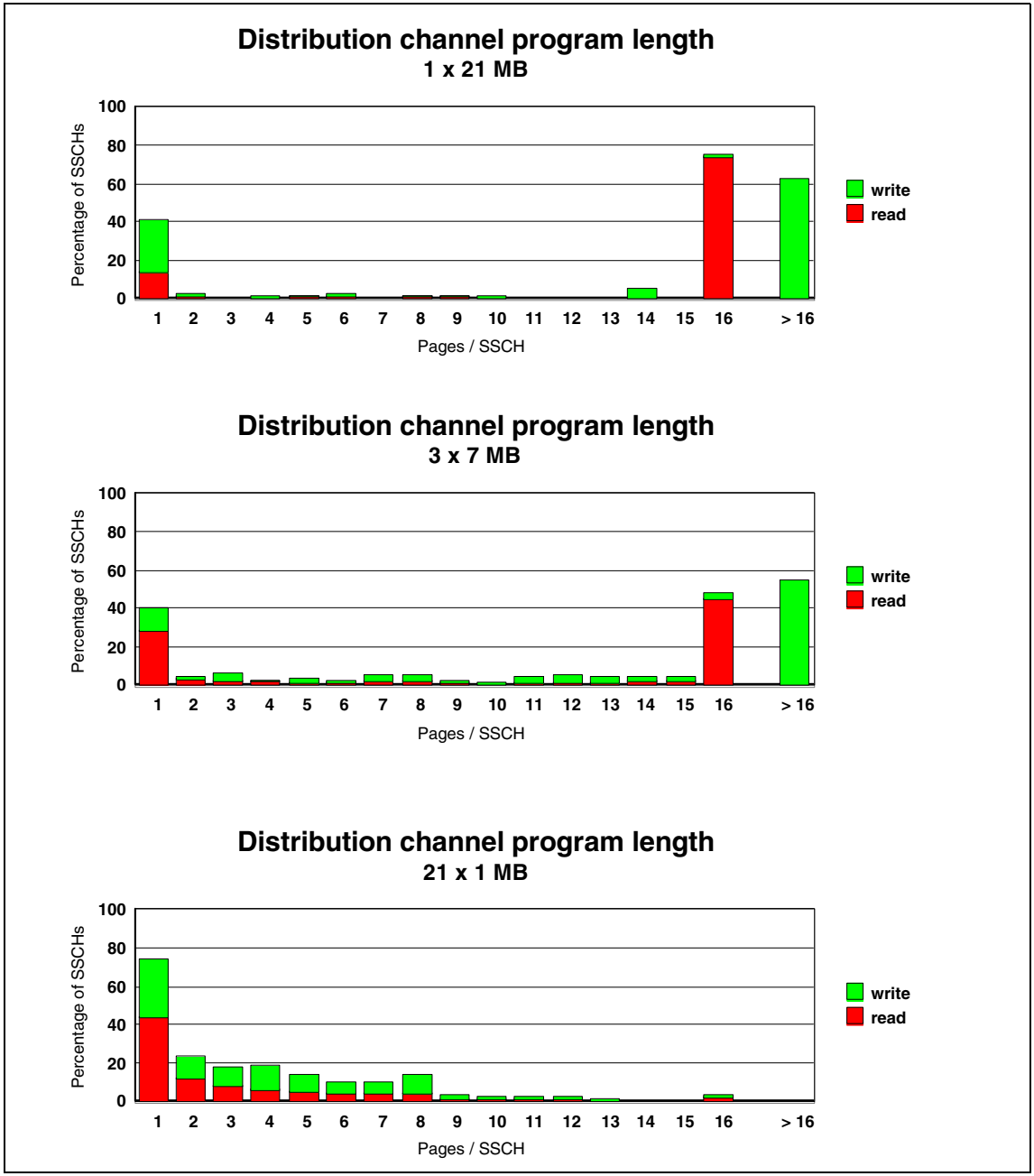


Figure 5-1 Distribution of channel program length

For the Test 1 x 21 test case, we see that swap-in typically is done either as single page I/O or with 16 pages in one I/O. For swap-out, the single page is popular, but more than half of the channel programs is for more than 16 pages (up to 128 pages per I/O).

With three processes running, we see more of the swap-in I/O gets shorter, and the peak at 16 pages per I/O for read starts to shrink.

Another increase in the number of processes drastically changes the picture. With 21 processes, there are hardly any swap-in I/Os of 16 pages. Both read and write channel programs have been reduced to less than 8 pages.

Looking at the length of the channel programs, we can also explain the increased MDC hit ratio, as reported in Table 5-2 on page 53. The VM minidisk cache was configured in so called “full track mode,” which means that MDC reads the full track from disk when the virtual machine wants to read something from the track. Subsequent reads for data from that same track can then be satisfied from MDC. Because the Linux dasd driver has no knowledge about the 3390 track geometry (or rather, chooses to ignore that information) the Linux I/O is not aligned on tracks (as z/OS normally does). On average, MDC will, therefore, read half a track of data more on each read that Linux does. With the short channel programs, it will frequently happen that the next read can completely be satisfied from the excess data read with the previous I/O by MDC. This counts as an MDC hit. The shorter the channel programs, the higher the MDC hit rate.

With the I/O trace from Example 5-7 on page 54, we can also determine which disk blocks are read and written. The difference in reference pattern is very obvious from the graphs in Figure 5-2 on page 57.

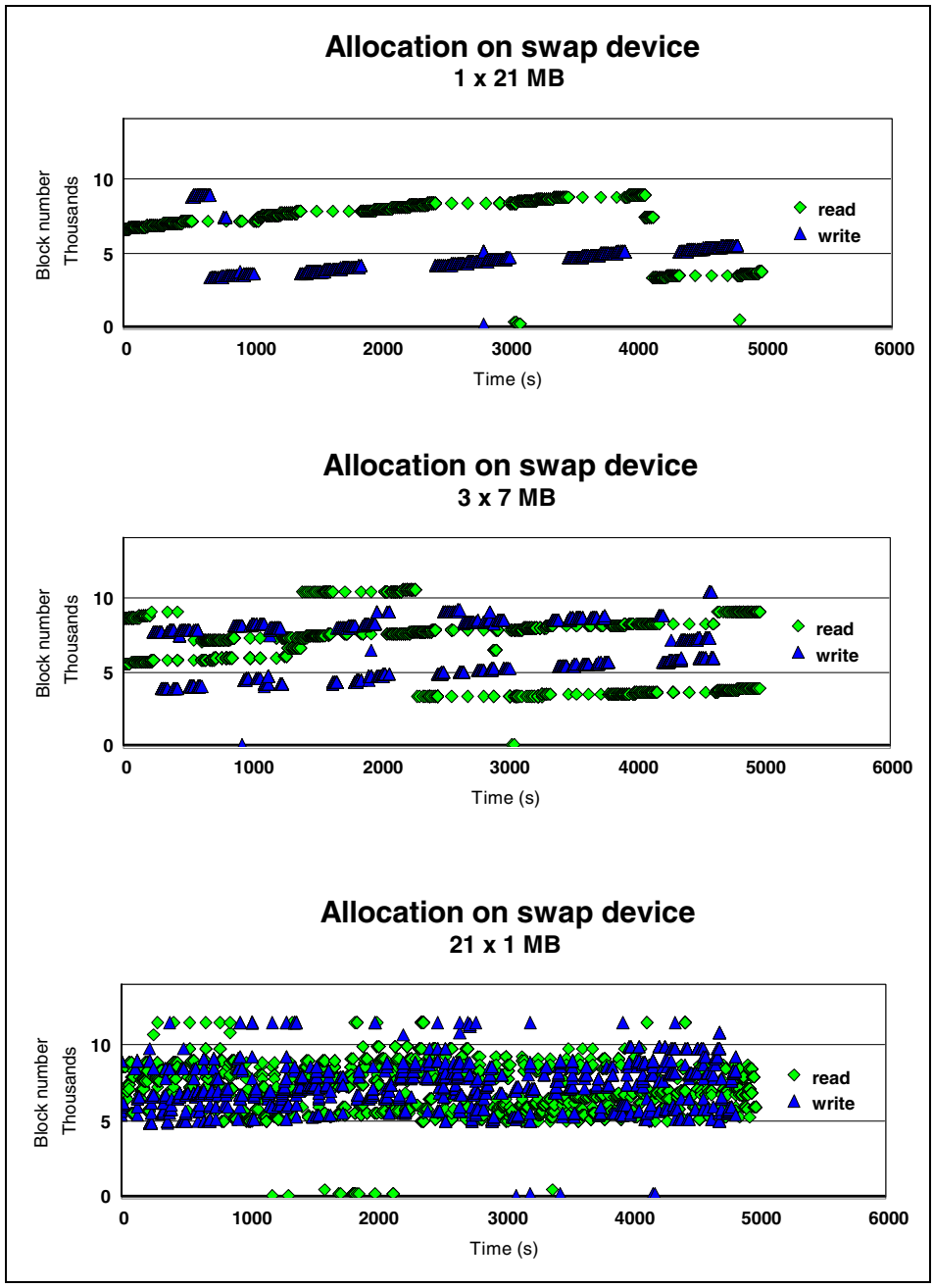


Figure 5-2 Swap device reference patterns

The first graphs clearly show that activity “sweeps” over the device. When a block must be read from disk, there is little choice, but when a new block is written to disk, Linux can pick any free spot on disk (with swapping to disk, every block written is a new block). That new spot is allocated in a way that increases the chances for Linux to write and read many consecutive blocks in a single I/O. To minimize seek times, the algorithm apparently prefers to write blocks close to where the last block was read (and where the “arm” is currently positioned). For the first two reference graphs, there is still a pattern visible.

The last graph in Figure 5-2 on page 57 looks rather chaotic. This is probably caused by the high number of parallel processes that all cause pages to be swapped out and in. With such a high degree of multiprogramming, the allocation strategy that Linux uses does not make it likely that blocks are read in the same order as they were written.

On modern S/390 DASD, the seek times are not much of an issue anymore. This makes the allocation strategy that Linux uses counter productive. Because the dasd driver uses SSCH to do the I/O, MDC is unable to decide whether data in cache remains valid. MDC, therefore, invalidates the tracks that are affected by the write CCWs. So by writing on the same tracks as where the read activity is happening, MDC is made less efficient.

5.2.2 Impact of page-cluster on MDC hit rate

The page-cluster value determines how many additional pages Linux will swap in on a page fault (under the assumption the process will want the next pages as well). Page-clustering is controlled by the /proc/sys/vm/page-cluster pseudo-variable.

In Example 5-8 on page 59, we illustrate the effect of page-clustering. Before starting the test, page-cluster size is set to one:

```
# echo 1 > /proc/sys/vm/page-cluster
```

Between 19:19:00 and 19:20:00, the page-cluster size is set to four (its default value):

```
# echo 4 > /proc/sys/vm/page-cluster
```

Note: Swap activity as reported by **`vmstat`** showed no change during this period.

Example 5-8 Changed page-cluster size and MDC hit rate

```
Screen: ESAUSR3  ITSO                          ESAMON V3.3  02/17 19:15-19:21
1 of 2  User Resource Utilization - Part 2     USER RMHTUX02        2064 COECB

                        DASD MDisk Virt Cache I/O    <---Virtual Device---->
         UserID    DASD Block Cache Disk   Hit Prty   <----I/O Requests----->
Time     /Class     I/O   I/O  Hits  I/O   Pct Queued  Cons   U/R  CTCA Other
-------- -------- ----- ----- ----- ---- -----  ----- ----- ----- ----- -----
19:21:00 RMHTUX02  7809     0  3792    0  48.6      0     0     0     0     0
19:20:00 RMHTUX02  7822     0  3770    0  48.2      0     3     0     0     0
19:19:00 RMHTUX02 16199     0 10018    0  61.8      0     0     0     0     0
19:18:00 RMHTUX02 15935     0  9881    0  62.0      0     0     0     0     0
19:17:00 RMHTUX02 15737     0  9937    0  63.1      0     0     0     0     0
19:16:00 RMHTUX02 16400     0 10245    0  62.5      0     0     0     0     0
```

We note in Example 5-8:

- A higher DASD I/O rate and corresponding MDC cache hit ratio in the interval 19:16:00 to 19:19:00. The page-clustering value in this interval is set to 1.
- The DASD I/O rate and MDC hit ratio decrease dramatically at 19:20:00. Beginning in this interval, the page-clustering value is returned to its default value of 4.

This test shows:

- **When the page-cluster value is small, Linux does many short I/Os.**
- **Entire tracks are read into the MDC, not simply pages.**
  The higher MDC hit ratio indicates that fetching an entire track from DASD anticipates subsequent I/O access to that DASD device.

5.3 The FBA discipline

Many people today suggest to use z/VM Virtual Disk (VDISK) as the swap device for Linux virtual machines. A VDISK is presented to the virtual machine as a fixed block architecture (FBA) DASD device (a virtual device of type 9336). Under the covers, the VDISK is an address space that lives in the z/VM main memory. The virtual machine issues I/O instructions against the device, and CP implements this by moving data between the virtual machine's primary address space and the VDISK address space.

Note: Ideally, one would expect that a real disk together with MDC could achieve the same effect. As shown in 5.2, “Swapping with ECKD discipline” on page 50, it does not work like that. MDC is a “write-through” cache, and the virtual I/O operation does not complete before the data is written to the real disk. Also, the data written to disk is not yet copied into MDC. Hence, MDC is only effective when data is read the second time after writing it.

5.3.1 Advantages of a VDISK swap device

The advantages of VDISK are that a very large swap area can be defined with very little expense. The VDISK is not allocated until the Linux server actually attempts to swap. Figure 5-3 illustrates the effect a 100 MB VDISK has on memory paging.

```
Screen: ESAVDSK  Velocity Software, Inc.        ESAMON V2.2  03/15 12:14-
                                              <--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
```

Figure 5-3 Paging with a 100 MB VDISK

The actual page cost of the VDISK amounted to 86 pages, of which 50 were paged out. When it was needed, more was allocated. This is much different than allocating a 100 MB real disk. Another large advantage of the virtual disk is that when the virtual disk is used, it becomes paged in and becomes a very fast “data in storage” type device.

5.3.2 Enabling an FBA VDISK

Because a VDISK appears as an FBA device to Linux, the dasd driver uses the "FBA discipline" by default for both SuSE and Red Hat distributions. When booting Linux with a 64 MB VDISK, the console messages in Example 5-9 show the FBA discipline is registered for the device.

Example 5-9 Booting Linux with FBA discipline

```
dasd(fba):FBA  discipline initializing
dasd(fba):0207 on sch 4: 9336/10(CU:6310/80) 64MB at(512 B/blk)
dasdh:(nonl)/        : dasdh dasdh1
dasd(fba):We are interested in: Dev 9336/00 @ CU 6310/00
dasd(fba):We are interested in: Dev 3370/00 @ CU 3880/00
dasd:Registered FBA discipline successfully
```

Just as with the ECKD swap device, the VDISK must be initialized with the **mkswap** command before the **swapon** command can use the device. Because the contents of the VDISK are not preserved after logoff of the virtual machine (the data on VDISK is volatile), the **mkswap** command must be issued each time you start the virtual machine.

There are at a few different ways to do this:

- Modify the init scripts in your Linux system to run the **mkswap** command early in the boot process. The disk can then be picked up automatically by the **swapon** command when Linux processes the /etc/fstab file. The **swapon** command can be issued manually if required.
- First IPL CMS in the Linux guest, and then use CMS tools to initialize the VDISK. This enables Linux to see the VDISK as a swap device just as if a **mkswap** command was already issued. If you need to use CMS to couple virtual channel-to-channels (CTCs), this may be a good option.
- Use some other virtual machine to link to all the VDISKs so that CP will retain them after the Linux guest is logged off. You still need some process to initialize the disks after a z/VM IPL. This approach is probably not very attractive because it will cause CP to retain more VDISKs than you need and put an additional burden on the paging subsystem.

The choice for initializing the disk in Linux or in CMS probably depends on the skills available and whether you are willing to change things in Linux. Neither the SuSE nor the Red Hat installers currently use VDISK as the swap device, but if you prepare the disk on CMS in advance, Linux will pick it up automatically.

Note: Writing to the raw VDISK in CMS requires serious programming. We show in 5.4.2, “Swapping with DIAGNOSE discipline” on page 65 that the Linux dasd driver picks up a CMS RESERVEd disk as well. Such a disk can be handled with normal CMS utilities. The RSRVDISK EXEC, shown is 5.9, “Initializing a VDISK using CMS tools” on page 74, is an example of a script to initialize a VDISK.

5.3.3 Swapping with FBA discipline

For the FBA VDISK, we did the same experiment as with the ECKD swap disk. Three hogmem processes were running, each allocating 7 MB of virtual storage (and we had to add one more for 1 MB to push it harder). Example 5-10 shows the reported swap rate from **vmstat**.

Example 5-10 The vmstat output of swapping to FBA disk

```
S vmstat 60|timestamp
18:49:00    procs                  memory      swap          io     system         cpu
          r  b  w   swpd   free   buff  cache   si   so    bi    bo   in    cs  us  sy  id
18:52:59 18  0  6  27260   1020     60    724 3889 4981  4057  4981    0   653  82  18   0
18:53:59 13  0 11  27228   2036     64    772 3802 4868  3970  4868    0   633  82  18   0
18:54:59 16  0  8  27228   1200     64    740 3713 4802  3868  4802    0   621  82  18   0
18:55:59 17  0  7  27228   1332     64    888 3719 4703  3874  4703    0   676  82  18   0
18:57:00 14  0 10  27236   1796     60    724 3741 4798  3900  4798    0   633  82  18   0
18:58:00 18  0  7  27232   1660     84    816 3635 4732  3786  4732    0   668  82  18   0
18:59:00 25  0  2  28316   1164     92   1264 4767 5422  4987  5424    0   833  79  21   0
```

We note the higher swap rate achieved for an FBA swap device in Example 5-10 compared to the corresponding swap rate for an ECKD swap device in Example 5-4 on page 50. The results are summarized in Table 5-4.

Table 5-4 Comparing swap rates for ECKD and FBA swap devices

	ECKD swap device	FBA swap device
Swap-in rate	1.5 MB/s	4 MB/s
Swap-out rate	1.8 MB/s	5 MB/s

The higher rate for FBA is expected. There is no physical device; instead, swapping occurs to z/VM memory.

Because no real device is involved, we can not look at device utilization. However, we can look at VDISK I/O rates, as shown in Example 5-11 on page 63.

Example 5-11 VDISK I/O rate while swapping (FBA discipline)

```
Screen: ESAUSR3  ITSO                          ESAMON V3.3  02/22 18:43-18:59
1 of 2  User Resource Utilization - Part 2     USER rmhtux02        2064 COECB

                        DASD MDisk Virt Cache I/O    <---Virtual Device---->
         UserID    DASD Block Cache Disk   Hit Prty   <----I/O Requests----->
Time     /Class     I/O   I/O  Hits  I/O   Pct Queued  Cons   U/R  CTCA Other
-------- -------- ----- ----- ----- ---- -----  ----- ----- ----- ----- -----
18:59:00 RMHTUX02 80388     0   837  79K   1.0      0     0     0     0     0
18:58:00 RMHTUX02 64389     0   622  63K   1.0      0     0     0     0     0
18:57:00 RMHTUX02 62614     0   534  62K   0.9      0     0     0     0     0
18:56:00 RMHTUX02 60930     0   487  60K   0.8      0     0     0     0     0
18:55:00 RMHTUX02 62138     0   474  61K   0.8      0     0     0     0     0
18:54:00 RMHTUX02 60323     0   532  59K   0.9      0     0     0     0     0
18:53:00 RMHTUX02 60866     0   518  60K   0.9      0     0     0     0     0
18:52:00 RMHTUX02 59806     0   443  59K   0.7      0     0     0     0     0
```

As explained in Example 5-5 on page 51, VDISK dose not use MDC. The reported MDC hits, therefore, apply to I/O other than the swap device (for example, to the root file system device). These I/O operations account for the difference in the `DASD I/O` and `Virt Disk I/O` columns.

The reported VDISK I/O rate during the period of approximately 75 K per minute (1250 I/Os per second) translates to some 7 KB data transferred per I/O operation.

Because of the rather short channel programs, the overhead of the swapping is rather high. This shows in the number of dispatches per second reported in Example 5-12 on page 64.

Example 5-12 Number of dispatches while swapping

```
Screen: ESAPLDV  ITSO                              ESAMON V3.3  02/22 18:55-19:00
1 of 2  Processor Local Dispatch Vector Activ   CPU ALL  USER rmhtux 2064 COECB

         <----Users----->   Tran       <VMDBK Moves/sec>   Dispatcher
Time     Logged Actv In Q   /sec  CPU  Steals  To Master   Long Paths
-------- ------ ---- *---  -----   --  ------  ---------   ----------
19:00:00    148  127 77.0  136.6   01   868.8         0.0       5230.4
                                     00     0.0         1.0       2590.9
18:59:00    148  127 70.0  135.7   01  1093.0         0.0       5419.6
                                     00     0.0         1.9       2121.2
18:58:00    148  125 75.0  146.5   01   654.0         0.0       3632.7
                                     00     0.0         1.2       3487.0
18:57:00    148  127 76.0  139.2   01   303.8         0.0       2162.3
                                     00     0.0         0.1       4949.6
18:56:00    148  125 76.0  134.2   01   501.2         0.0       2720.3
                                     00     0.0         0.7       4384.0
```

5.4 The DIAGNOSE discipline

As shown in Table 5-1 on page 48, by default, the Linux dasd driver uses the ECKD and FBA disciplines. In the past, the DIAGNOSE discipline had some bugs, and its use was not encouraged. The kernels shipped by SuSE and Red Hat, therefore, do not have the DIAGNOSE discipline built-in the kernel.

Restriction: For the Linux 2.4.7 kernel, the patches published on the IBM developerWorks® Linux for zSeries and S/390 home page are required to make the DIAGNOSE discipline work:

http://www-124.ibm.com/developerworks/oss/linux390/index.shtml

5.4.1 Using DIAGNOSE I/O for 3390 DASD

In Example 5-13 on page 65, we show how to enable the DIAGNOSE discipline for a swap device.

Example 5-13 Persuading Linux to use the DIAGNOSE discipline for the swap disk

```
S cat /proc/dasd/devices
0200(ECKD) at ( 94:  0) is dasda:active at blocksize: 4096, 36000 blocks, 140 MB
0201(ECKD) at ( 94:  4) is dasdb:active at blocksize: 4096, 180000 blocks, 703 MB
0202(none) at ( 94:  8) is dasdc:unknown                                          1
0203(none) at ( 94: 12) is dasdd:unknown
$ echo set 200 off > /proc/dasd/devices                                           2
$ modprobe dasd_diag_mod                                                          3
$ cat /proc/dasd/devices
0200(DIAG) at ( 94:  0) is dasda:active at blocksize: 4096, 36000 blocks, 140 MB
0201(ECKD) at ( 94:  4) is dasdb:active at blocksize: 4096, 180000 blocks, 703 MB
0202(none) at ( 94:  8) is dasdc:unknown                                          4
0203(none) at ( 94: 12) is dasdd:unknown
```

The process involves:

1. The 0200 disk is initially under control of the ECKD discipline of the dasd driver. Before starting the Linux system, this disk was formatted with CMS and RESERVEd.
2. The **echo** command instructs the dasd driver to stop using the device.
3. The DIAGNOSE discipline module (dasd_diag_mod.o) is loaded.
4. As part of its initialization, the DIAGNOSE driver looks for any disks it can handle. Because the 0200 disk was removed from the ECKD driver in step 2, the DIAGNOSE driver finds the disk immediately on initialization.

After loading the DIAGNOSE discipline module, the 0200 disk is under the control of the DIAGNOSE discipline, as seen in Example 5-14.

Example 5-14 Console messages when loading the DIAGNOSE discipline module

```
debug: unregistering dasda
dasd(diag):DIAG discipline initializing
dasd(diag):/dev/dasda (0200): capacity (4kB blks): 144000kB
dasda:CMS1/  SW0200(MDSK): dasda dasda1
```

5.4.2 Swapping with DIAGNOSE discipline

As with the other experiments, we also drive this system with three processes that allocate 7 MB each.

Example 5-15 Swapping three processes to disk using the DIAGNOSE discipline

```
14:40:54    procs                  memory      swap          io     system         cpu
14:40:54  r  b  w   swpd   free   buff  cache    si  so    bi    bo   in    cs  us  sy  id
14:40:54  3  0  2  25808   2036     60    736 2961 3015  3166  3017    0   206  94   6   0
14:41:54  2  1  2  25808   2036     64    784 2940 3112  3124  3112    0   203  94   6   0
14:42:54  3  0  1  25808   2036     64    928 3114 3136  3300  3136    0   207  94   7   0
14:43:54  3  0  1  25808   2036     68    748 3114 3127  3298  3127    0   209  94   6   0
14:44:54  3  0  1  25808   2036     80   1092 2982 3087  3165  3087    0   202  93   7   0
14:45:55  3  0  1  25808   1992     72    784 2962 3100  3183  3101    0   208  93   7   0
14:46:55  4  1  1  25808   2016     60    776 1848 2531  2004  2531    0   150  94   6   0
14:47:55  3  0  1  25808   2036     60    724 3102 3129  3288  3130    0   210  93   7   0
14:48:56  2  1  1  25808   2036     64    852 3068 3099  3249  3099    0   207  94   6   0
14:49:56  3  0  1  25808   2036     68    864 3001 3086  3179  3086    0   203  94   6   0
14:50:56  2  1  1  25808   2036     60    724 3059 3155  3279  3157    0   210  93   7   0
```

The reason we achieve a higher swap rate than when using the ECKD discipline can be contributed mainly to the MDC hit rate observed in Example 5-16.

Example 5-16 MDC hit rate with three processes: Diagnose discipline

```
Screen: ESAUSR3  ITSO                          ESAMON V3.3  02/24 14:40-14:51
1 of 2  User Resource Utilization - Part 2     USER rmhtux02        2064 C0ECB

                          DASD MDisk Virt Cache I/O    <---Virtual Device---->
          UserID    DASD Block Cache Disk   Hit Prty   <----I/O Requests----->
Time      /Class     I/O   I/O  Hits  I/O   Pct Queued  Cons   U/R  CTCA Other
-------- -------- ----- ----- ----- ---- ----- ----- ----- ----- ----- -----
14:51:00 RMHTUX02  8547  2850  5796    0 67.8      0     0     0     0     0
14:50:00 RMHTUX02  8229  2787  5571    0 67.7      0     0     0     0     0
14:49:00 RMHTUX02  8274  2694  5702    0 68.9      0     0     0     0     0
14:48:00 RMHTUX02  8391  2802  5721    0 68.2      0     0     0     0     0
14:47:00 RMHTUX02  6469  2783  3805    0 58.8      0     0     0     0     0
14:46:00 RMHTUX02  8464  2844  5727    0 67.7      0     0     0     0     0
14:45:00 RMHTUX02  8110  2744  5489    0 67.7      0     0     0     0     0
14:44:00 RMHTUX02  8453  2709  5868    0 69.4      0     0     0     0     0
14:43:00 RMHTUX02  8199  2661  5665    0 69.1      0     0     0     0     0
14:42:00 RMHTUX02  8210  2791  5543    0 67.5      0     0     0     0     0
14:41:00 RMHTUX02  8483  2769  5814    0 68.5      0     0     0     0     0
```

Even with the higher MDC hit ratio, the device is still fully utilized, as shown in Example 5-17 on page 67.

Example 5-17 Device utilization when swapping using the DIAGNOSE discipline

```
Screen: ESADSD6A ITSO                              ESAMON V3.3  02/24 14:40-14:51
1 of 3  DASD Performance Analysis - Part 2      DEVICE 3752          2064 COECB


          Dev        Device %Dev  SSCH  Resp  Serv <-Seek--> Read  Access
Time      No. Serial Type   Busy  /sec  Time  Time  Avg NonO  Pct Density
-------- ---- ------ ------ ---- ----- ----- ----- ---- ---- ---- -------
14:51:00 3752 LNXU4R 3390-3 85.0  49.1  18.7  17.3    0   23  0.0   17.31
14:50:00 3752 LNXU4R 3390-3 84.0  47.2  17.8  17.8    0    7  0.0   16.64
14:49:00 3752 LNXU4R 3390-3 82.2  45.0  20.1  18.3    0    7  0.0   15.84
14:48:00 3752 LNXU4R 3390-3 81.8  46.8  18.2  17.5    0    7  0.0   16.47
14:47:00 3752 LNXU4R 3390-3 68.8  47.2  14.6  14.6    0    7  0.0   16.63
14:46:00 3752 LNXU4R 3390-3 84.0  48.5  19.1  17.3    0   17  0.0   17.07
14:45:00 3752 LNXU4R 3390-3 81.9  45.8  18.6  17.9    0    6  0.0   16.16
14:44:00 3752 LNXU4R 3390-3 82.3  45.2  18.2  18.2    0    7  0.0   15.91
14:43:00 3752 LNXU4R 3390-3 84.0  45.2  20.4  18.6    0    7  0.0   15.92
14:42:00 3752 LNXU4R 3390-3 81.7  46.6  17.9  17.5    0    8  0.0   16.41
14:41:00 3752 LNXU4R 3390-3 83.5  47.3  18.7  17.6    0   15  0.0   16.67
```

Note: An effect that is not apparent from the proceeding example is the interactive response of the system. It is very bad. When typing a command, the echo of the input often takes seconds to show.

5.5 Using DIAGNOSE I/O for VDISK

The other option we have is use the DIAGNOSE discipline for a VDISK. After some debugging and reading the source code of the driver, we found that the DIAGNOSE discipline will only handle a VDISK when the CMS FORMAT is done with a block size of 512 bytes.

Note: Formatting with a 512 byte block size is not the default for the CMS FORMAT command. The small block size creates a lot of administrative overhead for the CMS file system. Using a small block size is not an obvious way to increase throughput.

In Example 5-18 on page 68, we see the console messages when loading a device under the DIAGNOSE discipline.

Example 5-18 Console messages when loading DIAGNOSE discipline for VDISK

```
dasd:/proc/dasd/devices: 'set 207 on'
dasd(diag):/dev/dasdh (0207): capacity (0kB blks): 65536kB
dasdh:CMS1/  SWP207(MDSK): dasdh dasdh1
```

5.5.1 Enabling DIAGNOSE I/O for VDISK

In Example 5-19, we show how to enable DIAGNOSE I/O for a VDISK.

Example 5-19 Persuading Linux to use DIAGNOSE discipline for VDISK

```
$ cat /proc/dasd/devices
0200(DIAG) at ( 94:  0) is dasda:active at blocksize: 4096, 36000 blocks, 140 MB
0201(ECKD) at ( 94:  4) is dasdb:active at blocksize: 4096, 180000 blocks, 703 MB
0202(none) at ( 94:  8) is dasdc:unknown
0203(none) at ( 94: 12) is dasdd:unknown
0204(none) at ( 94: 16) is dasde:unknown
0205(ECKD) at ( 94: 20) is dasdf:active at blocksize: 4096, 5400 blocks, 21 MB
0206(none) at ( 94: 24) is dasdg:unknown
0207(DIAG) at ( 94: 28) is dasdh:active at blocksize: 512, 131072 blocks, 64 MB          1
01cd(none) at ( 94: 32) is dasdi:unknown
$ mkswap /dev/dasdh1                                                                    2
Setting up swapspace version 1, size = 66539520 bytes
$ swapon /dev/dasdh1                                                                    3
$ swapoff /dev/dasda1                                                                   4
```

The process involves:

1. We verify the 0207 disk is using the DIAGNOSE discipline.
2. The **mkswap** command writes the “swap signature” to the swap device.
3. The **swapon** command adds the device to the list of swap devices, thus enabling Linux to use the device.
4. Finally, the **swapoff** command removes the original swap device (disk 0200) from the list of active swap devices. This command can take a moment to complete as Linux migrates swapped pages from this device to the active /dev/dasdh1 device.

5.5.2 Swapping with DIAGNOSE I/O for VDISK

With the VDISK controlled by the DIAGNOSE discipline of the driver, we seem to be able to get a much higher swap rate. However, we need to run more processes in order to stress Linux memory management enough to show this. We examine the swap rate in Example 5-20 on page 69.

Example 5-20 Swapping to VDISK through the DIAGNOSE discipline

```
r# vmstat 60|timestamp
19:54:00    procs                      memory        swap          io     system         cpu
19:54:00  r  b  w   swpd   free   buff  cache    si    so    bi    bo   in    cs  us  sy  id
19:54:00  1  0  0      0  16016    220   4472  1193  1158  1332  1159    0   256  44   3  53
19:55:00 18  0  3  25052   1720     44    720   330   501   413   505    0    65  85   2  13
19:56:00 18  0  7  25252   1712    112   1340   355   553   442   553    0    47  98   2   0
19:57:00 19  0  5  27228   1836     52    836  4458  5145  4698  5148    0   732  85  15   0
19:58:01 11  0 17  31624   1092     48    636 10278  8970 10631  8975    0  2010  74  26   0
19:59:01 12  0 18  31620   1028     56    752 10695  9449 10893  9449    0  2137  72  28   0
20:00:01  7  0 26  34328   2036    128   1764 12991 11472 13325 11478    0  2839  64  35   1
20:01:01 10  0 24  35952   1424     56    704 14398 12797 14701 12804    0  3311  60  39   1
20:02:01 10  0 25  38160   1072     52    580 18392 16289 18738 16292    0  4484  52  47   1
20:03:01 17  1 21  42468   1028     52    768 20113 17807 20461 17811    0  5089  47  52   1
20:04:01  7  0 40  51224   1020     32    572 26647 19170 27186 19177    0  6639  41  57   1
20:05:01  3  0 52  60040   1020     36    812 36314 18666 37115 18673    0  8729  35  63   2
20:06:01  4  0 50  60112   1016     32    724 43553 17260 44090 17262    0 10124  32  67   2
```

Note: This experiment shows a significantly higher swap-in rate than swap-out rate. On a stable system, one might expect each swap-out to be followed by a corresponding swap-in (and thus show equal rates). We believe the observed effect is due to multiple pages being brought in on a page fault, as discussed in 5.2.2, “Impact of page-cluster on MDC hit rate” on page 58.

When contention is high enough, many of the swap-in pages are stolen by other processes before the requesting process can access them. Because the swap-in pages are fresh (they have not yet been modified), they are excellent candidates to be stolen by Linux.

In Example 5-21 on page 70, we compare the VDISK swapping through the FBA driver to swapping through the DIAGNOSE driver.

Example 5-21 Comparing FBA and DIAGNOSE discipline VDISK I/O rates

I/O rate using DIAGNOSE driver

```
Screen: ESAUSR3  ITSO                              ESAMON V3.3  02/22 19:54-20:07
1 of 2  User Resource Utilization - Part 2         USER RMHTUX02        2064 COECB

                          DASD MDisk Virt Cache I/O    <---Virtual Device---->
         UserID    DASD Block Cache Disk   Hit Prty   <----I/O Requests----->
Time     /Class     I/O   I/O  Hits  I/O   Pct Queued  Cons   U/R  CTCA Other
-------- -------- ----- ----- ----- ---- -----  ----- ----- ----- ----- -----
20:05:00 RMHTUX02  2717     0  2604 239K  95.8      0     0     0     0     0
20:04:00 RMHTUX02  2133     0  2032 273K  95.3      0     0     0     0     0
20:03:00 RMHTUX02  1763     0  1704 278K  96.7      0     0     0     0     0
```

I/O rate using FBA driver

```
Screen: ESAUSR3  ITSO                              ESAMON V3.3  02/22 18:43-18:59
1 of 2  User Resource Utilization - Part 2         USER rmhtux02        2064 COECB

                          DASD MDisk Virt Cache I/O    <---Virtual Device---->
         UserID    DASD Block Cache Disk   Hit Prty   <----I/O Requests----->
Time     /Class     I/O   I/O  Hits  I/O   Pct Queued  Cons   U/R  CTCA Other
-------- -------- ----- ----- ----- ---- -----  ----- ----- ----- ----- -----
18:59:00 RMHTUX02 80388     0   837  79K   1.0      0     0     0     0     0
18:58:00 RMHTUX02 64389     0   622  63K   1.0      0     0     0     0     0
18:57:00 RMHTUX02 62614     0   534  62K   0.9      0     0     0     0     0
```

Note: The top half of Example 5-21 shows VDISK I/O rates using the DIAGNOSE driver; the bottom half reports I/O rates using the FBA driver (copied from Example 5-11 on page 63 and shown here for clarity).

Note the dramatic differences reported in `Virt Disk I/O` (273 K for DIAGNOSE versus 63 K for FBA). The difference in `DASD I/O` is accounted for by the fact that VDISK I/O when using FBA is counted as DASD I/O. When using DIAGNOSE however, VDISK I/O is not counted as DASD I/O; the reported numbers are the result of other I/O performed against the root device (memory contention causes portions of code such as libraries to be dropped from memory and later loaded back from disk).

Even though swapping to VDISK does not involve real disks, it is not free. Each page swapped by Linux must be copied by CP from the primary address space of the Linux virtual machine into the VDISK address space. CPU utilization reported in Example 5-22 on page 71 shows a large portion of the CPU cycles for the virtual machine are spent by CP on behalf of the user (the T/V ratio is high with 1.5).

Example 5-22 A high T/V ratio during heavy swapping to VDISK

```
Screen: ESAUSR2  ITSO                              ESAMON V3.3  02/22 20:00-20:08
1 of 3  User Resource Utilization                  USER rmhtux02        2064 C0ECB

                 <---CPU time--> <--------Main Storage (pages)--------->
         UserID  <(seconds)> T:V <Resident>  Lock <-----WSS----->
Time     /Class  Total  Virt Rat Total Activ  -ed Total Actv  Avg Resrvd
-------- ------- ----- ----- --- ----- ----- ---- ----- ---- ---- ------
20:08:00 RMHTUX02 59.78 38.57 1.5  6611  6611    8  6611 6611 6611      0
20:07:00 RMHTUX02 59.31 38.88 1.5  6611  6611    5  6611 6611 6611      0
20:06:00 RMHTUX02 59.66 39.64 1.5  6611  6611    0  6611 6611 6611      0
20:05:00 RMHTUX02 59.69 40.83 1.5  6611  6611    0  6611 6611 6611      0
20:04:00 RMHTUX02 59.60 42.78 1.4  6611  6611    0  6611 6611 6611      0
20:03:00 RMHTUX02 59.69 44.81 1.3  6611  6611    0  6611 6611 6611      0
20:02:00 RMHTUX02 59.76 46.22 1.3  6611  6611    0  6611 6611 6611      0
20:01:00 RMHTUX02 58.97 48.35 1.2  6611  6611    0  6611 6611 6611      0
```

Again, what the numbers do not show is the interactive behavior of the system. Unlike the other configurations, we find that with the DIAGNOSE discipline of the dasd driver swapping to VDISK, the interactive response continues to be good. Even with the system swapping at 40 MB/s, the interactive response is good enough to be editing a file without getting annoyed.

5.6 Using multiple VDISKs for swapping

Some recommendations for swapping involve the use of multiple swap partitions. When multiple swap partitions with the same priority are used, Linux will effectively spread the I/O over multiple disks and can achieve a higher I/O rate. When using VDISKs however, this does not apply. There is very little queueing for the VDISK, and any further increase of swapping is unlikely to provide any benefit.

There is, however, a good case for using multiple swap disk devices with different priorities. If multiple swap devices with different priority are available, Linux will attempt to fill the one with highest priority before using the next device. The algorithms for allocating pages on swap devices in Linux cause the active area to “sweep” over the device (this reduces seek times on the device and increases the ability of Linux to build long I/O chains).

This means that over time the entire swap device has been referenced. In the case of VDISK, this means that CP has to provide memory for the entire VDISK, even though the Linux virtual machine might only need a small portion of the VDISK at any given time. If memory contention is high enough, the contents of

the VDISK will be paged out by CP and must be brought back in when Linux next references that part of the VDISK.

If we give the Linux virtual machine two smaller VDISKs instead of one big VDISK and use them as swap device with different priorities, the “footprint” is effectively reduced by 50%.

5.7 Linux swap device recommendations

Excessive swapping in Linux is costly in both consumed CPU cycles and in response time. The best strategy is to reduce the amount of swapping that occurs in a Linux guest. Some general guidelines are:

- **Size the Linux virtual machine to reduce the amount of Linux swapping.**
  The optimum virtual machine size is a trade-off between reducing overall z/VM memory usage and reducing swapping in a Linux guest. As discussed in 3.5, “Conclusions for sizing z/VM Linux guests” on page 33, reduce the virtual machine size of Linux guest to the point where swapping begins under normal load, and then add an additional amount to minimize Linux swapping.
- **The amount of swap space to allocate depends on the memory requirements of your Linux guest.**
  The suggestion that swap space should be twice the memory size of a Linux machine should not apply to a z/VM Linux guest. If a Linux guest actually uses this much swap space, it probably indicates a larger virtual machine size should be allocated to the guest.
- **Do not enable MDC on Linux swap minidisks.**
  As stated in 3.1.2, “Linux swap space” on page 23, the read ratio is not high enough to overcome the write overhead.
- **Swapping to VDISK is faster than swapping to DASD.**
  In addition, when using a VDISK swap device, your z/VM performance management product can report swapping by a Linux guest. Be aware, however, that a VDISK is not recommended for z/VM memory-constrained systems, as discussed in 2.4.3, “VDISKs” on page 18.
- **The DIAGNOSE driver provides faster VDISK access than the default FBA driver.**
  Although DIAGNOSE discipline is faster, it requires more work to setup. Consider using the DIAGNOSE discipline you are comfortable with the additional effort to manage it.

- **Consider multiple swap devices rather than a single, large VDISK swap device.**
  Using multiple swap devices with different priorities can alleviate stress on the VM paging system when compared to a single, large VDISK. As discussed in 5.6, “Using multiple VDISKs for swapping” on page 71, a VDISK combined with a DASD swap device can provide a small, fast swap option (the VDISK) with spillover to a larger, slower DASD swap device.

5.8 Program text for hogmem

With hogmem, it is easy to run processes that allocate and use virtual memory. We use this to compare efficiency of the different swap devices in Linux. The program is shown in Example 5-23.

Example 5-23 Listing of hogmem.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <limits.h>
#include <signal.h>
#include <time.h>
#include <sys/times.h>

#define MB (1024 * 1024)

int nr, intsize, i, t;
clock_t st;
struct tms dummy;

void intr(int intnum)
{
    clock_t et = times(&dummy);

    printf("\nMemory speed: %.2f MB/sec\n", (2 * t * CLK_TCK * nr + (double) i
* CLK_TCK * intsize / MB) / (et - st));
    exit(EXIT_SUCCESS);
}

int main(int argc, char **argv)
{
    int max, nr_times, *area, c;

    setbuf(stdout, 0);
    signal(SIGINT, intr);
    signal(SIGTERM, intr);
    intsize = sizeof(int);
```

```
    if (argc < 2 || argc > 3) {
        fprintf(stderr, "Usage: hogmem <MB> [times]\n");
        exit(EXIT_FAILURE);
    }
    nr = atoi(argv[1]);
    if (argc == 3)
        nr_times = atoi(argv[2]);
    else
        nr_times = INT_MAX;
    area = malloc(nr * MB);
    max = nr * MB / intsize;
    st = times(&dummy);
    for (c = 0; c < nr_times; c++)
    {
        for (i = 0; i < max; i++)
            area[i]++;
        t++;
        putchar('.');
    }
    i = 0;
    intr(0);
    /* notreached */
    exit(EXIT_SUCCESS);
}
```

The program also reports a “memory bandwidth” in MB/s, but it should be clear that this has little value for Linux in a virtual machine.

5.9 Initializing a VDISK using CMS tools

The RSRVDISK EXEC, shown in Example 5-24 on page 75, can be used to initialize a VDISK in CMS before starting Linux. This is only sample code to illustrate what needs to be done. You should customize this script and call it from PROFILE EXEC.

Example 5-24 RSRVDISK EXEC: Make a VDISK into CMS RESERVEd

```
/* RSRVDISK EXEC     Format and Reserve a fresh VDISK               */
arg cuu .
if cuu = '' then signal usage

'PIPE (end \)',
  '\ command QUERY DISK',
  '| drop',
  '| spec 13 1',                       /* All used filemodes         */
  '| strliteral /ABCDEFGHIJKLMNOPQRSTUVWXYZ/',
  '| fblock 1',
  '| sort',
  '| unique single',
  '| take 1',
  '| append strliteral /*/',          /* A default                  */
  '| var fm'

if fm = '*' then call emsg 36, 'No free filemode found'
cuu = right(cuu,4,0)

queue '1'
queue 'SW'cuu
'FORMAT' cuu fm '( BLK 4K'
queue '1'
'RESERVE' userid() 'SWAP'cuu fm
'RELEASE' fm

return

emsg:
  parse arg rc, txt
  say txt
  if rc ¬= 0 then exit rc
return

usage: say 'SWAPDISK cuu'
```

After running the program, start Linux and initialize the disk with the **mkswap** command. This does not undo the CMS formatting, but only causes Linux to write some blocks in the “payload” of the disk (the single CMS file created on the disk). Then, without running the **swapon** command, shut down the Linux guest and IPL CMS again. Example 5-25 on page 76 shows how to copy the modified blocks.

Example 5-25 Copying the modified blocks from the RESERVEd disk

```
list * * K (date
FILENAME FILETYPE FM FORMAT LRECL       RECS     BLOCKS   DATE     TIME
RMHTUX02 SWAP0207 K6 F       4096      16360      16360  2/21/03  5:21:36
Ready; T=0.01/0.01 05:28:46
pipe diskrandom rmhtux02 swap0207 k number 1-16360 | strip trailing 00 | locate
11 | > sample swap0207 a | chop 10 | cons
         1
Ready; T=1.39/1.59 05:30:10
```

The PIPE command in Example 5-25 creates a small file on the A disk to hold the modified blocks (and it shows only a single block was modified). This file can be used to prepare a fresh VDISK again:

```
list * * c
RMHTUX02 SWAP0207 C6
Ready; T=0.01/0.01 05:47:57
pipe < sample swap0207 | pad 4106 00 | fileupdate rmhtux02 swap0207 c6
Ready; T=0.01/0.01 05:48:18
```

To Linux, that new VDISK, now looks just like the one that was prepared with the **mkswap** command before. If you would take the time to study the contents of the SAMPLE SWAP0207 file, you will find that it is easy to create the contents from scratch and even make it work for VDISKs of any size.

Tip: A similar process can be used if you want VDISK to hold temporary files for Linux. In this case, run the **mke2fs** command instead of the **mkswap** command against the device before you copy the payload from it. If necessary, you can also create directories (and even files) to have the disk in the correct state for when Linux boots.

6

CPU resources and the z/VM scheduler

In this chapter, we cover CPU resources and the z/VM scheduler. Topics include:

- Understanding LPAR weights and options
- The CP scheduler
- Virtual machine scheduling
- CP scheduler controls
- Analysis of the SET SRM LDUBUF control
- Virtual Machine Resource Manager

6.1 Understanding LPAR weights and options

There are two current trends:

- One is to consolidate multiple slower processors to much faster z/900s.
- The other is to separate workloads through the use of LPAR mode to avoid issues with the 2 GB line.

This section should clarify some of the configuration options.

Example 6-1 shows a logical partition (LPAR) report. Because z/VM often operates in multiple LPAR environments, understanding LPAR options and their impacts on performance will help you configure your systems to meet your business requirements.

Example 6-1 LPAR report

```
Report: ESALPAR  Logical Partition Analysis       ITSO Residency ESAMAP 3.3.0
Monitor initialized: on 2064 serial C0ECB First record analyzed: 01/30/03
------------------------------------------------------------------------------
         <--Complex--> <--Logical-> <--------Logical Processor---------->
         Phys Dispatch <-Partition> VCPU <%Assigned>          Cap-  Wait
Time     CPUs    Slice Name     No. Addr Total  Ovhd  Weight  ped   Comp
-------- ---- -------- -------- --- ---- -----  ----  ------  ----  ----
18:00:00   13  Dynamic A12       12    0  22.3   0.4      10    No    No
                                       1  35.7   0.4      10    No    No
                                          ----  ----
                                  LPAR  58.0   0.8
                      A1           1    0   5.3   0.5     180    No    No
                                        1   5.3   0.5     180    No    No
                                          -----  ----
                                  LPAR  10.6   0.9
                      A2           2    0   5.2   0.5      10    No    No
                                        1   5.2   0.5      10    No    No
                                          -----  ----
                                   LPAR  10.5   0.9
                      A3           3    0   5.3   0.5     180    No    No
                                        1   5.3   0.5     180    No    No
                                          -----  ----
                                  LPAR  10.6   0.9
                      A4           4    0   1.5   0.2      10    No    No
                                        1   0.3   0.0      10    No    No
                                          -----  ----
                                  LPAR   1.8   0.2
                      A5           5    0   3.2   0.3      10    No    No
                                        1   3.0   0.3      10    No    No
                                        2   2.9   0.3      10    No    No
                                        3   2.8   0.3      10    No    No
```

```
                          -----  ----
                     LPAR  11.8   1.2
A6          6    0   5.1   0.5      10    No    No
                 1   5.2   0.5      10    No    No
                          -----  ----
                     LPAR  10.3   0.9
A7          7    0   9.2   0.5      10    No    No
                 1   9.0   0.5      10    No    No
                          -----  ----
                     LPAR  18.3   1.0
A8          8    0   4.6   0.5      10    No    No
                 1   4.8   0.5      10    No    No
                          -----  ----
                     LPAR   9.4   0.9
A9          9    0   4.5   0.5      10    No    No
                 1   4.6   0.5      10    No    No
                          -----  ----
                     LPAR   9.1   0.9
A10        10    0   4.5   0.5      10    No    No
                 1   4.5   0.5      10    No    No
                          -----  ----
                     LPAR   9.0   1.0
A11        11    0   5.7   0.5     180    No    No
                 1   5.7   0.5     180    No    No
                          -----  ----
                     LPAR  11.4   0.9
C1         13    0 100.0   0.0     Ded    No   Yes
                 1 100.0   0.0     Ded    No   Yes
                          -----  ----
                     LPAR 200.0   0.0
C2         14    0 100.0   0.0     Ded    No   Yes
                 1 100.0   0.0     Ded    No   Yes
                           -----  ----
                     LPAR 200.0   0.0
C3         15    0 100.0   0.0     Ded    No   Yes
                 1 100.0   0.0     Ded    No   Yes
                          -----  ----
                     LPAR 200.0   0.1

System total logical partition busy:     770.7  10.8
```

In Example 6-1, the z/VM LPAR (A12) is listed first in the report. The report shows what z/VM considers its processor utilization to be. Assigned time is the time that a physical processor is assigned to a logical one. LPARs are prioritized by weights, which are explained in 6.1.2, “Converting weights to logical processor speed” on page 81.

All of the assigned times and utilizations shown in this report are in absolute numbers (as are the numbers reported in both ESAMAP and ESAMON), meaning these values are percentages of one processor. There is never a case of a reported “percent of a percent,” where one of the “percents” is not provided. Therefore, the VM LPAR is shown as using 58% of a possible 200% (two processors).

Other values reported for the A12 LPAR include:

- **Dispatch Slice** (set to `Dynamic`)
- **Capped** (set to `No`)
- **Wait completion** (set to `No`)

These are discussed further in 6.1.4, “LPAR options” on page 82.

There are two forms of overhead reported: logical and physical. Logical overhead can be charged to the LPAR, whereas physical overhead can not. There is a correlation between the number of logical processors defined and the amount of physical overhead involved in time slicing the physical processor between the logical processors: The *more* logical processors, the *higher* the overhead.

This example should be almost a worst case example. In this example, the logical overhead was 10.8% of one processor, but the physical overhead, as shown in the next section, was 39.9%.

6.1.1 Physical LPAR overhead

The overhead of managing the physical processors from Example 6-1 on page 78 is shown in Example 6-2 (the data comes from the same report as Example 6-1 on page 78).

Note: These reports show only standard processors (CPs); Integrated Facility for Linux (IFL) and Internal Coupling Facility (ICF) processors are not shown.

Example 6-2 LPAR physical overhead

```
Physical CPU Management time:
      CPU   Percent
      ---   -------
        0     6.815
        1     4.968
        2     5.000
        3     6.735
        4     4.874
        5     6.513
```

```
     6     4.920
     9     0.007
    10     0.007
    11     0.007
    12     0.008
    13     0.007
    14     0.007
           ______
Total:    39.870
```

From the report, we see there are seven shared physical processors (0-6); each with an overhead in the 5-6% range. The six dedicated processors (9-14) have less overhead. However, this does not imply processors should be dedicated to LPARs to reduce overhead. In this case, high overhead is caused by the large number of logical processors vying for time on the physical processors.

Note: To reduce physical overhead, use fewer LPARs and fewer logical processors.

6.1.2 Converting weights to logical processor speed

An LPAR is granted control of processors based on time slices. Each LPAR gets time slices based on the weighting factor for the LPAR. To determine the weight of each logical processor, use the following calculation:

1. **Add up all the weights of the logical processors.**
   In Example 6-1 on page 78, there are 12 LPARs (A1-A11) sharing seven logical processors (0-6) based on weighting. Of the 12, three have a weighted share of 180 (LPARs A1, A3, and A11). The remainder have a weighted share of 10. Therefore, the total weight is 630.
2. **Divide the weight of the "interesting LPAR" into the total.**
   This is the "logical share" of the physical complex allocated to the "interesting LPAR." LPAR A12 running z/VM has a weight of 10. Dividing this by the total shares (630) yields a 1.6% share of the seven shared processors.
3. **Divide the number of logical processors of the LPAR into the "logical share."**
   This is the share of each logical processor that is directly relative to the maximum speed at which a logical processor will operate if capped. Thus, 1.6% of seven processors is equivalent to about 10% of one processor ($1.6 \times 7 = 11.2$). Divide this into the two logical processors used by our VM system in LPAR A12, and each processor would be allocated 5% of one processor.

Note: This calculation is always applicable, even when the LPAR runs at less than 100% capacity. If an LPAR does not use its allocation, the extra CPU cycles are reallocated based on existing weights defined to other uncapped LPARs requesting more CPU. However, capped LPARs cannot acquire more CPU cycles than their assigned weight, even if those cycles are available.

With dynamic timeslicing, the LPAR weight is a guaranteed minimum, not a maximum allocation CPU resource. If all LPARs use their allotted share, this would be the amount of processing that could be performed. Normally (and in this case), very few of the LPARs had any activity. Thus, the A12 LPAR could get as much as 90% of each logical engine in its assigned time.

6.1.3 LPAR analysis example

For example, if the weight of an LPAR is 10, and the total weights of all LPARs is 1000, then the LPAR is allocated 1% of the system. If the system consists of 10 processors, the LPAR is allocated 10% of one physical processor. If the LPAR has two logical processors, each logical processor is allocated 5% of a physical processor. Thus, increasing the number of logical processors in a complex will decrease the relative speed of each logical processor in an LPAR.

6.1.4 LPAR options

LPAR shares can be defined as capped, meaning that their share of the physical system is capped to their given share. Given the situation of 1% allocated, this would be a very small amount of resource. If not capped, unused CPU resources are available to any LPAR that can utilize the resource based on given weights. Capped shares should *never* be used except in installations where a financial agreement exists to provide a specific speed of processor.

The time slice is either specific or dynamic. Specific time slices are rarely if ever used. The impact of having a specific time slice will likely mean erratic responsiveness from the processor subsystem. There does not seem to be any useful reason for using specific time slices.

Wait completion defines whether or not LPARs will either give up the processor when there is no remaining work, or keep it for the remaining time slice. With wait completion enabled, an LPAR will voluntarily relinquish the processor when its work is completed. This option may be useful for LPARs running as dedicated partitions.

6.1.5 Shared versus dedicated processors

When there are multiple physical processors on a system to be utilized by many different logical partitions, there is the option of dedicating processors to an LPAR. In general, this is only used for two reasons:

- For benchmarks to reduce questionable impacts from other workloads.
- When the workload is steady enough to utilize the processors, and there are sufficient resources to justify dedicated processors.

Important: When running benchmarks, always use dedicated processors to reduce the impact of other workloads on the results.

To justify the cost of zSeries implementations, the objective should always be high utilization. High utilization leverages the value of the reliability, availability, and serviceability of the zSeries systems. Other platforms rarely operate at high utilizations. Dedicating physical resources such as processors to one LPAR has the potential for reducing the overall system utilization. Reducing system utilization reduces zSeries effectiveness and increases the cost per unit of work.

6.2 The CP scheduler

The CP scheduler function attempts to keep as many of the logged-on virtual machines as possible operating concurrently. It takes into account the availability of processing time, paging resources, and real storage (as compared to virtual machine requirements).

The CP scheduler uses two time slices in determining how long a virtual machine competes for access to the processor:

- **Elapsed time slice**
  Virtual machines compete for use of the processor for the duration of the elapsed time slice.
- **Dispatch time slice**
  During its elapsed time slice, a virtual machine can only control the processor for a duration of its dispatch time slice. This is often referred to as the *minor time slice*.

When the dispatch time slice for a virtual machine expires, the scheduler readjusts its priority relative to other virtual machines competing for the processor. When its elapsed time slice expires, the scheduler drops the virtual machine from the set competing for the processor; the scheduler then attempts to add a virtual machine eligible for processor resources into the competing set.

6.2.1 Transaction classification

For dispatching and scheduling, virtual machines are classified according to their transaction characteristics and resource requirements:

- **Class 1**
  These virtual machines are designated as interactive tasks.
- **Class 2**
  These virtual machines are designated as non-interactive tasks.
- **Class 3**
  These virtual machines are designated as resource-intensive tasks.

A special class designation for virtual machines that require immediate access to processor resources. This class is referred to as Class 0.

For processor scheduling, started virtual machines reside on one of three lists:

- **Dormant list**
- **Eligible list**
- **Dispatch list**

6.2.2 The dormant list

The dormant list contains virtual machines with no immediate tasks that require processor servicing. As virtual machines require processor resources, they move to the eligible list.

6.2.3 The eligible list

The eligible list consists of virtual machines not currently being considered for dispatching due a system resource constraint (such as paging or storage). Virtual machines are kept in the eligible list when demand for system resource exceeds what is currently available. Virtual machines on the eligible list are classified according to their anticipated workloads requirements:

- **E1**
  E1 refers to Class 1 virtual machines expected to perform short transactions. Upon entering the eligible list for the first time, virtual machines are classified E1.
- **E2**
  E2 refers to Class 2 virtual machines expected to perform medium-length transactions. These virtual machines dropped to the eligible list after spending at least one elapsed time slice in E1 without completing processing.

- **E3**
  E3 refers to Class 3 virtual machines expected to perform long running transactions. E3 virtual machines spent at least two elapsed time slices on the eligible list without completing processing (at least one in E1 and one in E2).

Class 0 virtual machines do not wait in the eligible list for processor resources. Instead, they move immediately to the dispatch list. These are classified as E0 virtual machines. We discuss E0 virtual machines in 6.4.2, “The CP QUICKDSP option” on page 94.

As processor resources become available, virtual machines are moved from the eligible list to the dispatch list. Classification on the eligible list influences the priority and elapsed time slice assigned to virtual machines when they move to the dispatch list. Priorities assigned to virtual machines are intended to:

- Slow down virtual machines that are resource intensive in favor of virtual machines that require less resources.
- Ensure virtual machines receive a designated portion of the processor (see 6.4.3, “The CP SET SHARE command” on page 94).
- Control the amount and type of service based on virtual machine classification (E1, E2, or E3).

How the scheduler calculates priorities is discussed in 6.3.3, “Entering the dispatch list” on page 87.

6.2.4 The dispatch list

Virtual machines contending for processor time are placed on the dispatch list. Entries higher on this list are more likely to receive processor time. Virtual machines in the dispatch list retain the transaction classification assigned while waiting in the eligible list. Transaction classifications on the dispatch list are referred to as Q1, Q2, Q3, and Q0 (analogous to the E1, E2, E3, and E0 classification on the eligible list).

Note: E0 virtual machines on the eligible list are included in the count of Q0 virtual machines displayed by the CP INDICATE LOAD command.

6.3 Virtual machine scheduling

Figure 6-1 illustrates the state transitions involved in scheduling a virtual machine.

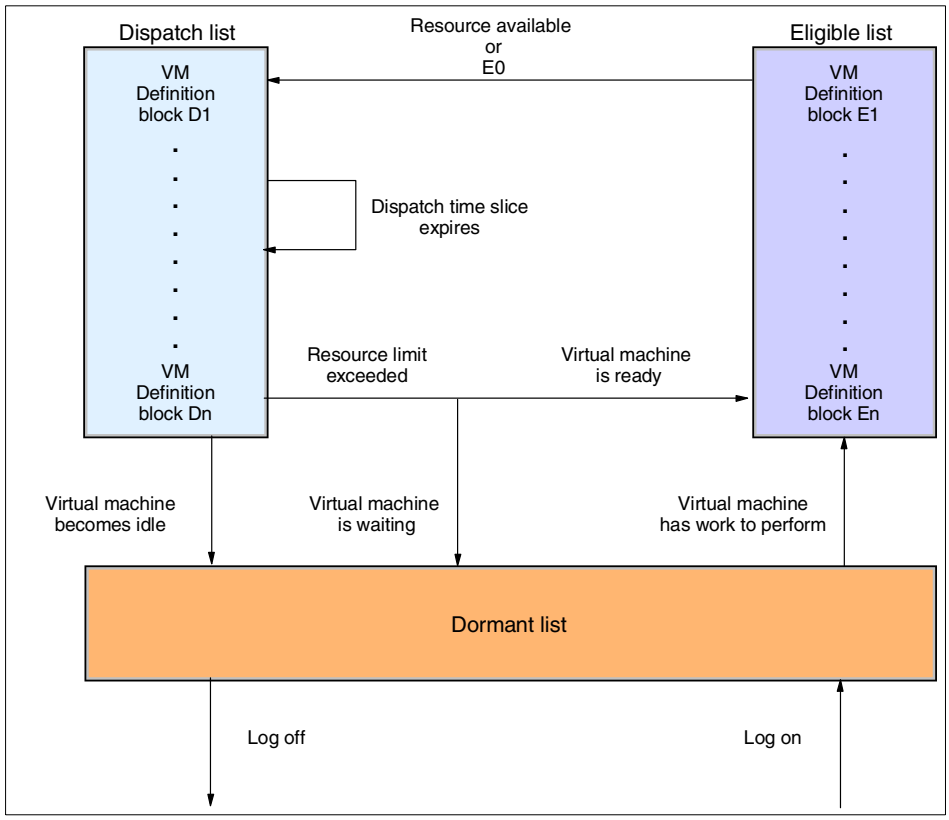


Figure 6-1 CP scheduling: State transitions

Definition: Figure 6-1 shows VM definition blocks on the dispatch and eligible lists. VM definition blocks represent virtual processors allocated to a virtual machine. By default, every virtual machine has at least one definition block (the base). Definition blocks are discussed in 6.3.4, “Scheduling virtual processors” on page 88.

6.3.1 Entering the dormant list

On logon, virtual machines are initially placed on the dormant list. Upon completing a transaction, virtual machines enter the dormant list from the dispatch list.

Note: This is referred to as “dropping from queue.”

A virtual machine can enter the dormant list from the dispatch if its elapsed time slice has expired and it is waiting for a resource (such as a demand page-in). When processor time is required, virtual machines are moved to the eligible list.

6.3.2 Entering the eligible list

On entry to the eligible list from the dormant list, virtual machines are normally classified as E1 (E0 virtual machines move directly to the dispatch list). Virtual machines can enter the eligible list from the dispatch list if they did not complete processing in their elapsed time slice. In this case, an E1 virtual machine drops to E2; E2 virtual machines drop to the E3; E3 virtual machines remain in E3.

6.3.3 Entering the dispatch list

In order to move from the eligible list to the dispatch list, a virtual machine must pass three tests:

- **Storage test**
  The virtual machine's working set must fit in real storage. We discuss this in "The STORBUF control" on page 90.
- **Paging test**
  If the user is a loading user, there must be room in the loading user buffer. Loading users are discussed in "The LDUBUF control" on page 90.
- **Processor test**
  There must be room in the dispatch buffer. The dispatch buffer size is discussed in "The DSPBUF control" on page 89.

A virtual machine on the dispatch list remains there for the duration of its elapsed elapsed time slice. Elapsed time slices are assigned to virtual machines as they enter the eligible list and are based on their transaction class.

CP assigns a initial elapsed time slice value of 1.2 seconds for E1 virtual machines during initialization. This value is dynamically adjusted during system operation:

- As the number of virtual machines that complete processing in E1 increases, the size of the E1 elapsed time slice decreases.
- As the number of E2 virtual machines increases, the size of the E1 elapsed time slice increases.
- If the number of E1 virtual machines drops below a threshold, and the number of E3 virtual machines increases above a threshold, the size of the E1 elapsed time slice increases.

The E1 elapsed time slice will always be a value between 50 ms and 16 seconds. E0, E2, and E3 elapsed time slices are assigned values 6, 8, and 48 times larger respectively than the E1 time slice.

The dispatch time slice is computed at CP initialization. Its value can be queried using the QUERY SRM DSPSLICE command.

While on the dispatch list, a virtual machine runs for its designated elapsed time slice. If processing has not completed in that period, it is moved to back to the eligible queue (with a lower workload classification if not already in the E3 queue).

6.3.4 Scheduling virtual processors

Additional VM definition blocks are created for each virtual processor defined to a virtual machine. These additional are linked to their respective base definition block. Both the base and any additional definition blocks cycle through the three scheduler lists. The base definition block owns the virtual storage and most virtual resources for the virtual machine. As the base and additional definition blocks move through the lists, the base is always in a list at least as high as any of its linked definition blocks.

Note: The hierarchy of lists is defined as (highest to lowest):

1. Dispatch list
2. Eligible list
3. Dormant list

This ensures the resource requirements of the virtual machine as a whole are considered when scheduling a VM definition block. However, processor time consumed by any additional definition blocks is measured independently of the base. This value is used when scheduling that block while on the dispatch list.

To illustrate, consider a virtual machine with two virtual processors (and two VM corresponding definition blocks), both of which start in the dormant list. If the additional definition block becomes dispatchable, both definition blocks (the base and the additional) move to the eligible list. After waiting in the eligible list, both definition blocks again move to the dispatch list and are given the same priority. As the additional definition block consumes resources however, its intermediate dispatch priority is calculated independently from the base block definition. We discuss virtual processors in more detail in 7.6, “Performance effect of virtual processors” on page 120.

6.3.5 z/VM scheduling and the Linux timer patch

Linux servers that run a timer to do some small amount of work every 10 ms break the z/VM scheduler model. With the 10 ms timer interrupt, CP classifies a Linux virtual machine as a long-running task and assigns it to Q3.

Using the “on-demand” timer patch available from the IBM developerWorks site corrects this situation. With this patch applied, Linux virtual machines are dispatched less frequently and are less resource intensive. This can reduce the number of Q3 servers competing for resources. Reducing the concurrent number of tasks competing for resources reduces the contention felt by the shorter tasks. We examine this effect in 7.3, “The Linux timer patch” on page 108.

6.4 CP scheduler controls

The CP scheduler can be influenced using two general types of controls:

- SRM controls globally influence the overall processor resources.
- Local controls influence how an individual virtual machine is regarded by the scheduler.

6.4.1 Global SRM controls

Overall z/VM processor resources can be tuned using the CP SET SRM command.

The DSPBUF control

The CP SET SRM DSPBUF command controls the number of users in the dispatch list. It permits you to overcommit or under-commit processor and I/O device resources. The command format is:

```
SET SRM DSPBUF i j k
```

Where:

i Specifies the number of dispatch list slots available for E1, E2, and E3 users.

j Specifies the number of dispatch list slots available for E2 and E3 users.

k Specifies the number of dispatch list slots available for E3 users.

Note: Valid operand ranges are $32767 \geq i \geq j \geq k \geq 1$.

For example:

```
SET SRM DSPBUF 35 30 18
```

This command allocates 35 slots on the dispatch list:

- Five are guaranteed to E1 users ($35 - 30$).
- 30 are available to E2 and E3 users, 12 of which ($30 - 18$) are guaranteed not to be occupied by E3 users.
- 18 are available to E3 users (although these may be occupied by E1 and E2 users).

Important: The DSPBUF control is a risky knob! We do not recommend making adjustments to this control.

The LDUBUF control

The CP SET SRM LDUBUF command partitions the commitment of the system's paging resources.

Definition: LDUBUF stands for loading user buffer. A loading user is a heavy user of paging resources and is expected to have a high paging rate. A loading user is defined as a user that takes five page faults within one time slice of work. Because a time slice is relatively very small, any user that takes five page faults within that time is totally consuming the equivalent of one paging device. Use the INDICATE QUEUES EXP command to display the current loading users.

The command format is:

```
SET SRM LDUBUF i j k
```

Where:

i Specifies the percentage of system paging resources available to E1, E2, and E3 users.

j Specifies the percentage of system paging resources available to E2 and E3 users.

k Specifies the percentage of system paging resources available to E3 users.

The STORBUF control

The CP SET SRM STORBUF command partitions the commitment of real storage in terms of pages based on the transaction class (E1, E2, E3) of a user. This command enables you to overcommit or under-commit real storage.

The command format is:

```
SET SRM STORBUF i j k
```

Where:

i Specifies the maximum percentage of system storage available to E1, E2, and E3 users.

j Specifies the maximum percentage of system storage available to E2 and E3 users.

k Specifies the maximum percentage of system storage available to E3 users.

Note: Valid operand ranges are $999 \geq i \geq j \geq k \geq 0$. The default values for STORBUF are 125%, 105%, and 95%, respectively.

Performance gains might be realized by overcommitting real storage when expanded storage is available. When you overcommit storage this way, a virtual machine's working set is still computed as before, but the apparent real storage available to virtual machines who want to enter the dispatch list appears larger. Therefore, more virtual machines are allowed into the dispatch list. This might result in higher paging rates, but often the benefit of reducing the eligible list and moving users into the dispatch list will offset this increase.

The CP scheduler perceives storage requirement for virtual machines that do not drop from queue to be much larger than actual. Because the scheduler cannot determine the real storage requirements, in this case, raising the STORBUF control can improve system performance. This effectively disables the storage test discussed in 6.3.3, "Entering the dispatch list" on page 87.

Linux guests currently do not drop from queue due to:

- **The effect of the Linux timer**
  (discussed in 7.3, "The Linux timer patch" on page 108)
- **QDIO network devices**
  (discussed in 7.4, "QDIO and the dispatch queue" on page 112)

For Linux guests, setting the STORBUF control is an inexact science. There is little difference in a STORBUF value of 900, as opposed to a value of 300; in both cases, the storage test is likely disabled.

The MAXWSS control

The CP SET SRM MAXWSS command sets the maximum working set a normal user on the dispatch list is allowed to have. If the user's working set size exceeds

this percentage, that user is dropped back into the eligible list from the dispatch list.

The command format is:

```
SET SRM MAXWSS m
```

Where:

m Specifies the maximum percentage of system pageable storage available to a user.

MAXWSS is intended to prevent large virtual machines from acquiring an inordinate amount of system memory at the expense of smaller users.

Note: In order for this parameter to work as intended, virtual machines must actually reside on the eligible list (that is to say, the scheduler must consider the system to be storage constrained based on the STORBUF settings).

The DSPSLICE control

A virtual machine is assigned a dispatch time slice each time it is dispatched. The dispatch time slice is calculated at system initialization based on real processor speed and represents a fixed number of instructions processed. This value can be changed using the CP SET SRM DSPSLICE command:

```
SET SRM DSPSLICE min
```

Where:

min Specifies the minimum dispatching time slice (in milliseconds).

Note: Valid dispatching time slice values are in the range $100 \geq min \geq 1$.

When a virtual machine is first dispatched, it runs until:

- Its minor time slice interval expires.
- It completes the workload and proceeds to wait for more work.
- It enters CP mode.
- An interrupt occurs.

When redispatched, the virtual machine is assigned a new dispatch time slice (if it relinquished the processor due to an interrupt, it is assigned the remaining portion of the previous dispatch time slice).

When to use SRM controls

SRM controls can be used to ensure:

- Processor resources are utilized as much as possible.
- Thrashing situations do not occur.
- When performance is really bad (as *will* happen), some servers get preferential access to processor resources.

For example, if performance is very bad, you would still want TCP/IP (or perhaps a DNS server or security manager) to perform well. Thus, it is preferable to utilize the SRM controls properly and not overuse the QUICKDSP option.

Important: The QUICKDSP option should be used for machines that need to run when things are very bad. Set QUICKDSP to the select few that must absolutely perform well.

You should use STORBUF to over-allocate storage when virtual machines do not drop from queue. In this situation, the dispatching and scheduling algorithms become skewed. The true storage requirement is unknown, and proper scheduling is impossible. Installations found that when running many servers (even “idle servers” that never drop from queue), the perceived storage requirement is very high. Using high STORBUF values to “over-allocate” storage can improve this situation.

In an installation where the virtual machines do drop from queue, using STORBUF to over-allocate storage might not be appropriate.

Note: In order to drop from queue, Linux guests must run with the timer patch installed (see 6.3.5, “z/VM scheduling and the Linux timer patch” on page 89) *and* use an appropriate network architecture (see 7.4, “QDIO and the dispatch queue” on page 112).

For this case, if there is expanded storage, the SET SRM XSTORE command should be set to at least 50%, or even to its maximum value.

In environments where there is considerable paging, paging should be controlled using the proper SET SRM LDUBUF command. The default LDUBUF in a paging environment allows the paging subsystem to be 100% consumed (see 6.5, “Analysis of the SET SRM LDUBUF control” on page 95). At the point where the paging subsystem is at 100% utilization, intuitively, there would not be much value at increasing the LDUBUF and allowing more users to consume paging resources.

In 6.5, “Analysis of the SET SRM LDUBUF control” on page 95, we examine the operation of some SRM controls.

6.4.2 The CP QUICKDSP option

The SET QUICKDSP command does one thing only, but it does that very well. It should be used for service machines that are required when there are serious memory or processor constraints.

The CP scheduler classifies virtual machines assigned the QUICKDSP option as E0 virtual machines. An E0 virtual machine is added immediately to the dispatch list whenever it has work to do, without waiting in the eligible list. The CP scheduler bypasses the normal storage, paging, and processor tests when moving E0 virtual machines from the eligible list to the dispatch list.

6.4.3 The CP SET SHARE command

One of the most misunderstood (and most misused performance options) is the SHARE value of a virtual machine. The first choice of SHARE is to use ABSOLUTE or RELATIVE. The second choice is the size of the SHARE.

The simplest way to decide which to use for a specific server is to answer the question: As more users log on to this system, should this service machine get more CPU or less?

- A relative share says this server should get a relative share of the processor, relative to all virtual machines in the dispatch and eligible lists. As more users log on, the share will drop.
- Absolute shares remain fixed up to the point where the sum of the absolute shares is 100% or more, a rather confused state of configuration.

Servers such as TCP/IP or even DNS servers might have a requirement that rises as the level of work increases. These servers should have ABSOLUTE shares. All other users should use RELATIVE.

Size of share is both a business decision and a performance decision. For example, if one server is assigned a very high share, which might be as much of the system as the rest combined, one would expect this server to be absolutely critical to your business. This server has the capability of taking resource whenever it needs it, and if this server starts looping, would easily consume all the resource allocated.

A simple way of looking at share values: If there is heavy contention for the processor, what servers would you like to run.

TCP/IP is obvious, as are required services such as Domain Name System servers. TCP/IP and other such required servers should have absolute values that approach the CPU consumption that would be required at peak loads, *never* more than that.

6.5 Analysis of the SET SRM LDUBUF control

The purpose of the SET SRM LDUBUF command is to control thrashing. Thrashing is a situation where paging is at a point where less work is being accomplished because of contention for paging and storage devices. The following analysis shows some of the key points in evaluating the use of LDUBUF.

When the loading capacity is evaluated by the scheduler, the number of paging devices is the “loading capacity.” Setting LDUBUF to the default of `100 85 65` allows 100% of the total capacity (the number of paging devices) to be utilized. Thus, if there were seven paging devices, seven users that were considered “loading” would be allowed onto the dispatch list. Additional users would be delayed on the eligible list until one or more of the existing loading users either dropped from the list or obtained their working set in storage and stopped taking page faults.

For this analysis, a benchmark was developed with 100 servers that would allocate storage, perform a function, and release the storage, typical of many Linux environments. What the measurements show is that LDUBUF at the default level will allow the total paging subsystem to be 100% consumed.

6.5.1 Default setting analysis

Example 6-3 on page 96 shows CPU utilization and paging rates using the default settings for LDUBUF and STORBUF.

Note: The paging DASD devices are RAMAC Virtual Array (RVA) units.

Example 6-3 CPU and paging with default LDUBUF/STORBUF settings

```
Report: ESASSUM      Subsystem Activity                              Velo
Monitor initialized:                      on 2064 serial C0ECB     Firs
------------------------------------------------------------------------

         <---Users---->  <Processor> Storage (MB) <-Paging--> <-----I/O
         <-avg number->  Utilization Fixed Active <pages/sec> <-DASD-->
Time       On Actv In Q  Total Virt.  User Resid. XStore DASD Rate Resp
-------- ---- ---- ----  ----- ----- ----- ------ ------ ---- ---- ----
17:26:00  154  130 85.0    121    81 134.6 2874.7    523 5025  400 22.9
17:27:00  154  133 71.0    106    60 132.7 2870.5    245 5027  408 22.4
17:28:00  154  129 78.0     96    49 134.2 2870.1    225 5269  415 19.8
17:29:00  154  129 74.0    161   127 133.4 2872.3    299 5546  540 15.4
17:30:00  154  129 83.0    152   123 133.5 2871.0    302 5572  576 14.7
17:31:00  154  129 82.0    108    87 134.7 2871.7    417 5431  599 13.7
17:32:00  154  132 82.0    116   100 134.1 2875.1    445 5335  763 11.2
17:33:00  154  128 81.0    115   100 134.2 2870.4    478 5241  858 10.2
17:34:00  154  130 43.0    135   121 132.6 2877.8    615 4671  929  9.3
17:35:00  154   87 59.0    109    99 132.4 2879.9   1006 3615  789  9.2
```

The report shows CPU utilization varied from 96% to 161% with a paging rate to DASD of over 5000 pages/second. The paging to expanded storage was 400-500 pages/second.

6.5.2 User queue analysis

Example 6-4 on page 97 shows a queue analysis for a benchmark class of users named IUCVRO. These users normally drop from queue when idle, as they use the IUCV driver and have the timer patch applied (as discussed in 7.4, “QDIO and the dispatch queue” on page 112).

Example 6-4 User queue analysis

```
Report: ESAUSRQ      User Queue and Load Analysis
Monitor initialized:                       on 2064
-------------------------------------------------
         <-----------User Load------------>
UserID   Logged  Non-          Disc- Total  Tran
/Class       on  Idle  Active  conn  InQue  /min
-------- ------ ----- ------- ------ -----  ----
17:26:00  154.0     .   130.0      .  85.0  4590
Hi-Freq:  154.0   130   130.0    144  93.2  4715
 ***User Class Analysis***
*Servers   18.0     5     5.0     15   2.1  46.0
*Keys       4.0     3     3.0      4   2.1  10.0
*TheUsrs   14.0     5     5.0      8   2.0  79.0
IUCVRO     99.0    99    99.0     99  67.9  4580
REDHAT      9.0     9     9.0      9   9.0     0
SUSE31      5.0     5     5.0      5   6.0     0
SUSE64      5.0     4     4.0      4   4.0     0
```

Note that at this peak time, of the 99 logged on, 99 are performing some amount of work each minute, making them all “active” and “Non-idle.” Out of the 99, on average 67.9 (68) are in queue. One other important note: Service machines that drop from queue are identified as having transactions. Note that the other Linux servers do not have transactions; they do not drop from queue.

6.5.3 DASD analysis

Example 6-5 on page 98 shows an analysis of direct access storage device (DASD) during the default run.

Example 6-5 Analysis of paging devices

```
------------------------------------------------------
Report: ESADSD2      DASD Performance Analysis
Monitor initialized:                      on 2064 ser
------------------------------------------------------

 Dev        Device <--SSCH--> <%DevBusy> <SSCH/sec->
 No. Serial Type   Total  ERP  Avg  Peak   avg  peak
---- ------ ------ ------ ---  ---- ---- ----- -----
17:26:00
***Top DASD by Device busy***
1590 430PG5 3390-3   1864   0  98.5 98.5  31.1  31.1
15D0 430PG6 3390-3   2051   0  98.5 98.5  34.2  34.2
1551 430PG8 3390-3   1987   0  98.1 98.1  33.1  33.1
1550 430PG4 3390-3   1915   0  98.1 98.1  31.9  31.9
1511 430PG7 3390-3   1831   0  97.9 97.9  30.5  30.5
3753 430PAG 3390-3   2848   0  97.9 97.9  47.5  47.5
3B44 430PG2 3390-3   4058   0  97.4 97.4  67.6  67.6
1512 LX1512 3390-3   1150   0  26.1 26.1  19.2  19.2
1552 LX1552 3390-3   1217   0  24.5 24.5  20.3  20.3
15D2 LX15D2 3390-3    828   0  20.9 20.9  13.8  13.8
***End Top DASD by Device busy***
```

Notice the seven page devices that show up in the top 10 device list on the DASD Performance report. These devices are 98% busy and can *not* perform any additional work. It should be obvious that allowing more loading users into the queue will not accomplish any additional work. In this case, it would be better to keep some of these users on the eligible list.

Example 6-6 on page 99 shows an analysis of the channel subsystem.

Example 6-6 Analysis of channel subsystem

```
Report: ESADSD1      DASD Configuration
Monitor initialized:
Monitor period:        13080 seconds (
----------------------------------------
 Dev Sys         Device    <CHPIDS OnLn>
 No. ID   Serial Type   SHR 01 02 03 04
---- ---- ------ ------ --- -- -- -- --
1511 088D 430PG7 3390-3 NO  42 57 24 4B
1512 088E LX1512 3390-3 NO  42 57 24 4B
1550 08CC 430PG4 3390-3 NO  42 57 24 4B
1551 08CD 430PG8 3390-3 NO  42 57 24 4B
1552 08CE LX1552 3390-3 NO  42 57 24 4B
1590 090C 430PG5 3390-3 NO  42 57 24 4B
15D0 094C 430PG6 3390-3 NO  42 57 24 4B
15D1 094D LX15D1 3390-3 NO  42 57 24 4B
15D2 094E LX15D2 3390-3 NO  42 57 24 4B
3750 0F0A 430RES 3390-3 NO  1B 27 32 3D
3753 0F0D 430PAG 3390-3 NO  1B 27 32 3D
3B44 12FE 430PG2 3390-3 NO  41 4C 36 56
```

If adding page devices were an option, there should be sufficient channel capacity to support additional devices. In this DASD configuration, several paging devices share four channel paths (CHPIDs). Five paging devices share CHPIDs 42, 57, 24, and 4B.

Now that we know what channel paths are being used, we evaluate the capacity of the channels being utilized in Example 6-7.

Example 6-7 Analysis of channel capacity

```
Report: ESACHAN      Channel Performance Ana
Monitor initialized:                     on
-----------------------------------------
         <Pct Channel>           <extended->
Time/     Utilization        CMG Utilization
CHPID     LPAR  Total  Shrd Type LPAR  TOTAL
--------  ----  -----  ---- ---- ----- -----
24           .   93.3  Yes     1  86.9  86.9
42           .   90.0  Yes     1  84.9  85.7
4B           .   85.0  Yes     1  85.6  85.7
57           .   93.3  Yes     1  88.4  89.0
```

There are different measurements provided by the monitor. This report shows three different values:

- The `<Pct Channel> Utilization Total` number is a sampled value. Sampling is performed by CP at the default high-frequency sample rate of 1 sample/second. In this case, CP sampled the channels and found them about 90% busy.
- The `<extended>` reported values are provided by the I/O processor. These measurement provide values at both the `LPAR` and `TOTAL` levels. If multiple LPARs were sharing channel paths, these values could be different (due to the amount of time the CHPIDs were in use by other LPARs).

Because of the differences in data sources, there will be some differences. In this case, all values show the channel paths at above 85%. This is *too* high to support more devices and is a performance issue. To reduce this utilization, paging must be reduced or more channel paths utilized.

6.6 Virtual Machine Resource Manager

Virtual Machine Resource Manager (VMRM) is a new facility available in z/VM 4.3. It runs in a service virtual machine (VMRMSVM) and dynamically manages the performance of *workloads*. VMRM uses a set of user-specified workload definitions and goals, compares these with the achieved performance, and makes adjustments accordingly. (It is conceptually somewhat similar to the Workload Manager used by z/OS.) The basic idea is to allow performance objectives to be set in a manner more closely aligned with business objectives than has been the case previously.

VMRM is only effective in a constrained environment because it works by prioritizing workloads in order to enhance their access to resources, and accordingly restricts other work's access to those resources. If the z/VM system

is not constrained, VMRM can't enhance access to resources because they are readily available anyway.

A *workload* is a collection of one or more virtual machines that are to be treated as an entity for the purpose of performance management. Each workload has certain goals defined for it. These goals are for DASD and CPU utilization. A workload can have a DASD goal, a CPU goal, or both. A goal represents the relative importance of a workload for the particular installation. Configuration changes can be made at any time, but require the VMRM service machine to be stopped and restarted to make them effective.

VMRM uses CP MONITOR data to determine the level of activity in the system and whether or not workloads are meeting the defined goals. VMRM allows some latitude, 5%, in making this determination (this helps avoid overreaction when a workload is near its goals).

Every minute (the default) VMRM examines the system and workloads and picks a workload that is failing to meet its goals. If it finds one, it then uses the CP SET SHARE and CP SET IOPRIORITY commands to adjust the workload to give it more access to the resource or resources for which the goal or goals were not met. VMRM remembers which workloads it adjusted recently and does not alter them again for a while. Instead, it might choose another workload to adjust at the next interval.

VMRM cannot adjust users with fixed CPU or I/O priorities. If you have defined users with absolute or hard limited SHARE, or absolute I/O priority, this will have to be changed for VMRM to manage them. This also allows specific users that otherwise would be treated as part of a workload to be effectively excluded from it, but it would normally be more sensible not to define such a user as part of a workload in the first place.

6.6.1 Implications of VMRM

Because VMRM sits there adjusting performance parameters at frequent intervals, it is able to adjust to changing conditions. As the workload on the z/VM system changes over the course of a day, VMRM can make adjustments to the defined (and running) workloads in order to try to make them meet the goals defined for them. (It cannot ensure workloads meet their goals because it cannot create resources out of thin air.)

Because VMRM adjusts z/VM tuning parameters, it might conflict with manual efforts to tune z/VM. In particular manual use of the CP SET SHARE and CP SET IOPRIORITY commands is likely to cause problems, or at least unclear results.

Support for I/O priority queueing was added to z/VM to support VMRM. This is probably as significant as the addition of VMRM itself.

CP SET IOPRIORITY and the associated directory statement were added to z/VM to support VMRM. However, these can also be used for manual tuning instead.

Because VMRM uses CP MONITOR data, it might be affected by other users of the CP MONITOR data, and vice versa. As is always the case when more than one thing uses CP MONITOR data, some care in setup is required.

As a result of its dynamic nature, VMRM makes benchmarking difficult. Benchmarking typically requires repeatable conditions, and the continual adjustments of VMRM make repeatable conditions unlikely.

VMRM is not directly influenced by memory usage and has no direct effect on memory usage.

6.6.2 Further information about VMRM

For further information about VMRM, refer to *z/VM V4R3.0 Performance*, SC24-5999. VMRM concepts are introduced in *z/VM, VSE, and Linux Technical Conference foils: z/VM Resource Management*, by Christine Casey, available at:

http://www.vm.ibm.com:2003/pdfs/V612up.pdf

7

Tuning processor performance for z/VM Linux guests

This chapter discusses tuning processor performance for Linux guests. Topics include:

- Processor tuning recommendations
- The effect of idle servers on performance
- The Linux timer patch
- QDIO and the dispatch queue
- Infrastructure cost
- Performance effect of virtual processors

7.1 Processor tuning recommendations

CPU time is limited to the number available processors. Steps should be taken to reduce processor requirements:

- **Eliminate unnecessary Linux services.**
  Default Linux guest installations typically start services that probably are not used. These services consume CPU cycles even if no useful work is performed. Remove unneeded services from the Linux startup sequence.
- **Remove unneeded cron-initiated tasks.**
  Look for and remove unneeded tasks started by **`cron`**.
- **Reduce processor usage by idle Linux guests.**
  Ensure idle Linux guest do not consume unnecessary processor resources. Thing to consider are:
  - **Install the timer patch.**
    Waking the Linux scheduler 100 times per second wastes processor resources.
  - **Eliminate “are you there” pings.**
    Do not ping an idle guest simply to verify that it is alive.
  - **Do not measure performance on idle guests.**
    Measuring an idle guest costs processor resources.
- **Consider infrastructure processing requirements.**
  Consider the cost of various infrastructure configurations, such as:
  - **Router configuration**
    Routing costs are discussed in Chapter 9, “Measuring the cost of OSA, Linux, and z/VM networking” on page 143.
  - **Installation and cloning costs**
    See 7.5.2, “Installing new systems” on page 115 for a comparison of cost of installing Linux guests.
- **Prioritize workloads.**
  When there are processor constraints, use share options to determine what work gets done.

More performance tips can be found on at Velocity Software’s Web site at:

`http://linuxvm.com`

7.1.1 Processor performance on a constrained system

When the processor is constrained, the other resources will be underutilized. There is not much point in tuning other subsystems when the processor is already overcommitted. When the processor is constrained, all options to reduce

processor requirements should be evaluated. We discuss some options in 7.6, “Performance effect of virtual processors” on page 120.

7.2 The effect of idle servers on performance

In a shared resource environment, there is no room for unnecessary processes. Servers that run cron jobs for historical reasons should be redesigned.

Under z/VM, as a shared resource environment, it is not optimal to wake up servers to make sure the servers are active, or to monitor them to query current activity. In this environment, the virtual machines will be tailored for optimal performance and having unneeded interrupts or work being performed detracts from resources available for productive work.

In a virtual environment, one of the most expensive use of resources is waking up an idle server simply to perform a trivial task. It is important to apply the Linux timer patch in order to reduce resource usage by idle Linux guests (see 6.3.5, “z/VM scheduling and the Linux timer patch” on page 89).

Pinging a server to see if it is alive keeps the server active. This uses resources that are likely better utilized by other servers performing real work. This expense should *always* be monitored and minimized.

Two common mistakes made in a virtual server infrastructure is to ping applications to ensure they are “alive” and to monitor the performance of the virtual servers. When idle, the servers would normally not take significant amounts of either storage or processor resource. But when either the applications are pinged or the servers are monitored, the server must have resident storage to provide the appropriate positive responses. A better approach in this virtual environment for monitoring is to utilize the existing and mostly free (in terms of resource requirements) VM monitor.

By default, most Linux distributions start several services that might not be needed. To illustrate the cost of these services, we show the CPU usage and DASD I/Os in Figure 7-1 on page 106.

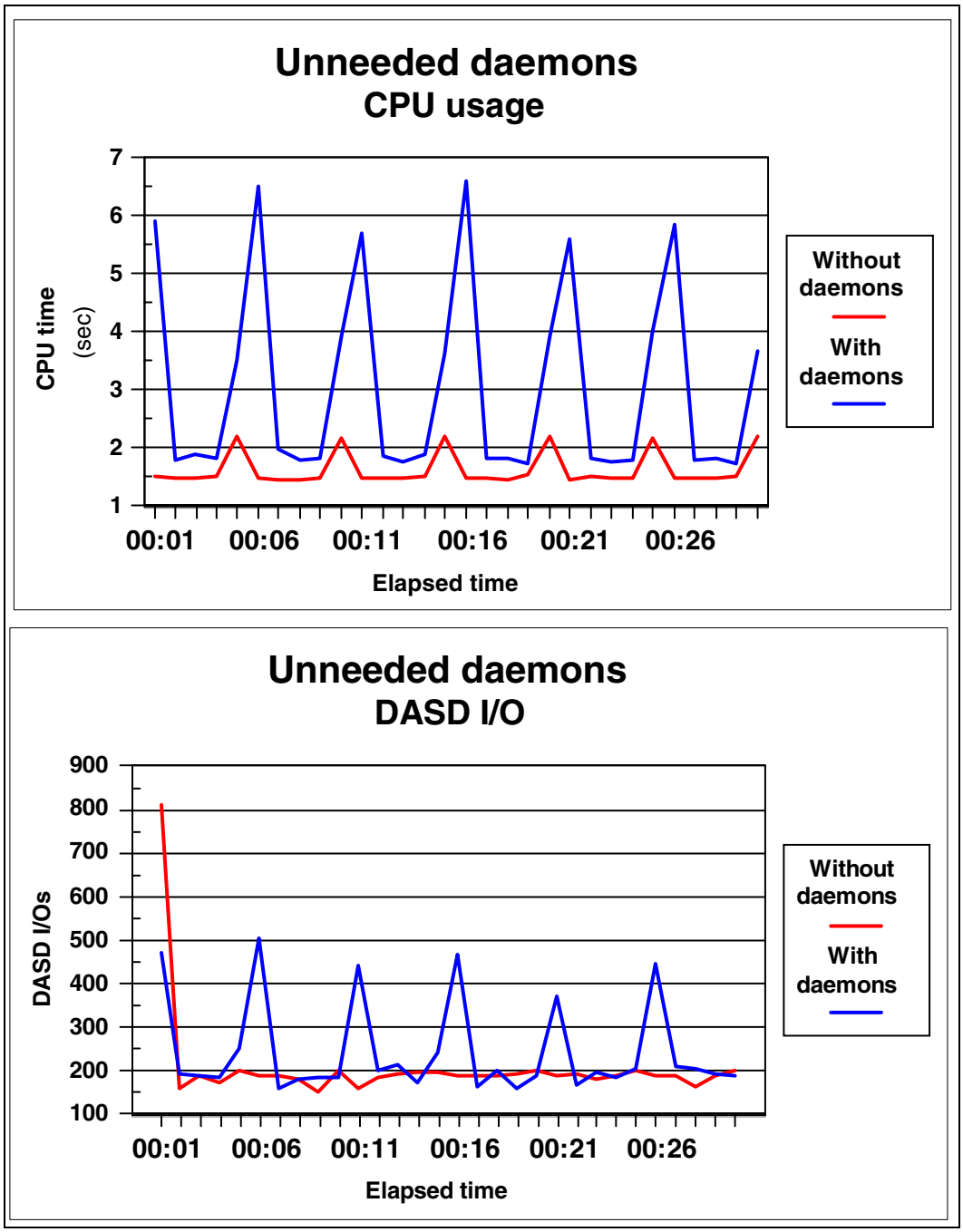


Figure 7-1 The effect of unneeded daemons on CPU and DASD usage

Services that might be considered unnecessary in a Linux guest include:

- **sendmail**
  If receiving or delivering mail is not required on the Linux guest, consider stopping the sendmail server.
- **anacron**, **atd**, and **cron**
  These daemons are responsible initiating tasks at regular intervals. If no useful tasks are automatically scheduled, stop these services.
- **autofs**, **nfs**, **nfslock**, and **portmap**
  The autofs daemon is responsible for automatically mounting file systems; nfs and nfslock provide Network File System (NFS) support; portmap provides Remote Procedure Call (RPC) services required by NFS.
- **lpd** and **xfs**
  Printing services are provided by the lpd daemon; xfs provides X-Windows fonts.
- **inetd/xinetd**
  The inetd (or the replacement xinetd) daemon manages Internet services such as FTP and Telnet.

7.2.1 Network Time Protocol daemon

Linux on z/VM has some problems with keeping track of time. There are two reasons for this:

- **We believe there is a bug in the kernel that causes the Linux daytime to drift in periods of high activity.**
  This should be fixed, but it might need some time to convince people of the exact cause. The “on-demand timer” patch was reworked for the 2.4.17 kernel; we are not sure whether this kernel exhibits the same problem.
- **Although the zSeries time-of-day (TOD) clock does not drift much, many people set the clock manually at IPL (using a watch as reference).**
  This is not an exact way to set the clock, and the effect is that the z/VM time has a constant offset to the official time. Even if Linux would use the TOD clock to tell time, it would be different from external guests using NTP for it. This might impact distributed applications (for example, because time stamps on shared files are wrong).

A suggestion has been to fix both of these issues by running **`ntpd`** on each Linux guest. This will talk to NTP servers and adjust the Linux clock continuously to keep it synchronized to the rest of the world. The disadvantage of this approach is that it is very expensive to do so. A process wakes up every second to adjust the clock, and this keeps the server busy all the time.

If the first problem was fixed, all we need is something to set the time initially when the guest is IPLed. We must be able to use the z/VM command SET VTOD for this to synchronize the virtual TOD clock of a Linux guest with the virtual TOD clock of one other virtual machine that synchronized (once after IPL) with NTP servers outside.

Note: We did not have time to implement this solution, so we can only speculate on what performance improvement it might yield.

7.3 The Linux timer patch

Traditionally, the Linux kernel keeps track of time using a timer that interrupts the kernel at a constant rate. On each interrupt, the global variable “jiffies” is incremented and various queues inspected for work. On dedicated hardware, this has a minimal performance effect. However, when running Linux running on z/VM, this is not a good idea.

Although this method of keeping track of time may not be the most efficient, the biggest problem for Linux on z/VM is that the Linux guest uses a little bit of CPU cycles every 10 ms. This causes the z/VM scheduler to keep the guest “in queue” (and therefore, unused memory pages in the Linux virtual machine cannot be trimmed, as discussed in 2.3, “z/VM use of memory” on page 13). With many Linux guests holding on to their working sets, z/VM memory will fill up rather quickly.

This problem is addressed with the so called “on-demand timer” patch that can be downloaded from the IBM developerWorks Linux for zSeries and S/390 home page:

`http://www-124.ibm.com/developerworks/oss/linux390/index.shtml`

This patch does away with the 10 ms timer tick and sets the timer to pop only when the kernel needs to wake up. Even though this does not make a real “zero load idle Linux guest,” the periods between two timer ticks are normally long enough for z/VM to recognize the guest as idle (and start taking measures to trim memory pages).

Note: With the 2.4.19 kernel (as used in SuSE SLES 8), the on-demand timer is part of the main source. It is enabled by writing to the /proc/sys/kernel/hz_timer pseudo variable.

The VM/RTM output in Example 7-1 on page 109 shows two virtual machines:

- RMHTUX01 runs a kernel with the timer patch applied.

- LNXR09 runs an unpatched kernel.

The %CPU column shows the Linux machine with the on-demand timer uses less CPU resources.

Example 7-1 Two Linux guests, one with timer patch applied

```
<USERID> %CPU %CP %EM ISEC PAG  WSS  RES   UR PGES SHARE VMSIZE TYP,CHR,STAT

LNXR09    .62 .06 .56  .20 .00  28K  34K   .0    0   100   128M VUX,DSC,DISP
RMHTUX01  .01 .00 .01  .00 .00  23K  24K   .0    0   100   128M VUX,DSC,DISP
```

Even though saving CPU resources is very welcome, the real big benefit of the changed timer behavior is that z/VM is able to recognize that the virtual machine as idle. When a virtual machine does not consume resources for more than 300 ms, the z/VM scheduler assumes that a transaction has ended and that the virtual machine went into a long-term wait. At that point, z/VM will start to page out some of the working set of the idle virtual machine when main memory is constrained. When the virtual machine becomes busy again (even if only after a second or less), the page fault handling will page in necessary portions of the virtual machine's memory. This reduces the footprint of the idle Linux virtual machine.

Note: Even with the on-demand timer, an idle Linux virtual machine is still reported as active by VM because it wakes up frequently. Be careful when using the terms "idle" and "active" in this context. The idle Linux machine probably behaves more like an interactive CMS user.

When a Linux guest with the on-demand timer still uses a lot of CPU time when idle, that is normally caused by some process or kernel thread requesting frequent wakeup calls. In some cases, these frequent wakeup calls are part of the application design, and in other cases, it is simply a bug.

Important: In the Linux 2.4.7 kernel, a bug in ReiserFS causes the virtual machine to wake up every 50 ms. Even though this is not as bad 10 ms, it is not long enough for z/VM to recognize transactions. We believe it is safe to assume this is a bug, because later kernels cause the ReiserFS thread to wake up every five seconds. We can argue whether a wakeup every five seconds is the best way to implement that particular function, but the damage is certainly less than with 50 ms.

7.3.1 Analyzing the timer ticks

When the on-demand timer patch is applied to the kernel, the number of timer interrupts goes down. In some systems, the number of timer interrupts stays rather high. If the cause is not obvious, you can look at the number of timer ticks and the process requesting them.

A simple way to count the number of timer ticks is to run a TRACE EXT in the Linux virtual machine for a fixed period of time and look at the output. Example 7-2 illustrates a program executed from a privileged user ID (class C) to count timer ticks.

Example 7-2 Sample program (COUNTEXT EXEC) to count timer ticks

```
/* COUNTEXT EXEC     Trace timer interrupts to count them          */
arg uid ; if uid = '' then exit 24
cmd = 'CP SEND CP' uid

cmd 'SPOOL PRT PURGE'
'CP SLEEP 1 SEC'
cmd 'TRACE EXT 1004 PRINTER RUN'
'CP SLEEP 60 SEC'
cmd 'SPOOL PRT' userid() 'CLOSE'
'CP SLEEP 1 SEC'
cmd 'TRACE END ALL'
```

The SLEEP command in the program lets the target Linux virtual machine run for 60 seconds and closes the printer spool file after that time. We execute the script in Example 7-3.

Example 7-3 Executing COUNTEXT EXEC

```
countext rmhtux02
RDR FILE 0180 SENT FROM RMHTUX02 PRT WAS 0018 RECS 0083 CPY
```

The spool file arrival message indicates TRACE wrote 83 records to the spool file in the one minute interval the script was sleeping. Expect this number to vary between tests.

When the number of timer ticks is higher than expected, we trace the process requesting the wakeup calls. A good address to trace is the entry point of the schedule_timeout() function in the kernel. You can its address from the current System.map file on your system. Alternatively, you can use /proc/ksyms, as shown in Example 7-4 on page 111.

Example 7-4 Determining the address of schedule_timeout

```
# cat /proc/ksyms | grep schedule_timeout
0010ae74 schedule_timeout
```

Start the trace from the 3270 console; if you use the **hcp** command through a Telnet session, you create work on the Linux guest that obscures the measurements. The following CP TRACE command prints the value of current on entry to the schedule_timeout() function:

```
#CP TRACE I R 10AE74.2 TERM RUN CMD D C40.4
```

Example 7-5 shows the trace output.

Example 7-5 Tracing the value of current at entry to schedule_timeout()

```
-> 0010AE74'  STM   908FF020 >> 005C3EB8    CC 0
V00000C40  005C4000                          06 L00000C40
 -> 0010AE74' STM   908FF020 >> 01F49E30    CC 0
V00000C40  01F4A000                          06 L00000C40
 -> 0010AE74' STM   908FF020 >> 005C3EB8    CC 0
V00000C40  005C4000                          06 L00000C40
 -> 0010AE74' STM   908FF020 >> 005C3EB8    CC 0
V00000C40  005C4000                          06 L00000C40
```

Tip: The current variable identifies the process running when the call to schedule_timeout() is made. The task_struct for the process is located 8192 (0x2000) bytes before current. The process identifier (PID) is located at offset 0x70 into the task_struct.

Using the sort stage of *CMS Pipelines,* a simple pipe gives the breakdown per process for our one-minute interval. Using **grep** against the output of **ps -ef**, we identify the process name.

We can further enhance the trace to show which timer the process sets. From the source code and the generated S/390 assembler instructions, we find that register R2 contains the length of the requested delay in jiffies. Using this information, we are able to produce Table 7-1 on page 112 to identify which processes are responsible for generating timer ticks.

Table 7-1 Breakdown of one minute Linux timers

Address	Jiffies	Count	PID	
005C4000	100	60	5	[kswapd]
005EC000	500	12	1	init
01F4A000	201	30	173	/usr/sbin/nscd
01F46000	1501	4	174	/usr/sbin/nscd
01F7C000	1501	4	167	/usr/sbin/nscd
0201A000	6001	1	153	/usr/sbin/cron
02024000	2147483647	2	139	/sbin/slogd
03FFE000	500	12	8	[kupdated]

As seen in the table, the kswapd kernel process causes Linux to wake up every second. This kernel thread monitors the Linux memory usage, waking up every second to adjust counters and checking if memory pages need to be swapped out (as discussed in 3.2.1, “Page cleaning” on page 24). One could argue that on an idle system, there is little reason for the daemon to wake up because nothing has changed in the system.

The nscd processes requesting wakeup calls every two to 15 seconds stand out (nscd caches user and group information; however, this is not needed when using flat files in /etc directory of a local disk). This process is started in the default SuSE installation. We had not removed it from the startup sequence.

Note: The work done at each wakeup call is minimal. For the 113 wakeup calls per minute, our system uses 110 ms of CPU time, or approximately 0.1% of a single CPU. When we stopped the nscd daemons, the number of timer interrupts went down to 87 per minute. CPU usage dropped to 20 ms per minute, or 0.02% of a CPU.

7.4 QDIO and the dispatch queue

We found that some Linux guests stay in the dispatch queue (Q3, in fact) even with the “demand timer” active (see 7.3, “The Linux timer patch” on page 108). The VM/RTM output shown in Example 7-6 on page 113 shows two virtual machines (RMHTUX01 and RMHTUX02) with the timer patch applied. For comparison, the LNXR09 Linux guest does not have the timer patch.

Example 7-6 Three Linux guests, two with the timer patch applied

```
<USERID> %CPU %CP %EM ISEC PAG  WSS  RES   UR PGES SHARE VMSIZE TYP,CHR,STAT

LNXR09    .62 .06 .56  .20 .00  28K  34K   .0    0   100   128M VUX,DSC,DISP
RMHTUX01  .01 .00 .01  .00 .00  23K  24K   .0    0   100   128M VUX,DSC,DISP
RMHTUX02  .01 .00 .01  .00 .00  17K  17K   .0    0   100   128M VUX,IAB,DISP
```

The %CPU column shows that the Linux guests with the on-demand timer indeed use far less CPU resources.

The IAB status for RMHTUX02 indicates a difference from RMHTUX01 (although both have the on-demand timer patch). z/VM considers RMHTUX02 to be an interactive user. In Example 7-7, the output of an INDICATE QUEUES command confirms RMHTUX01 is in Q3 while RMHTUX02 is not.

Example 7-7 INDICATE QUEUES showing timer patch guest in Q3

```
CP IND QUEUE
RSCS          Q3 PS  00001224/00001223 RMHTUX01      Q3 PS  00023149/00022411
LNXR06        Q3 PS  00007003/00005923 LNXR04        Q3 PS  00007008/00005976
...
LNXR05        Q3 PS  00007008/00005880 LNXR02        Q3 PS  00007015/00005851
VCOLLECT      Q0 PS  00000910/00000889 ESATCP         Q2 PS  00000584/00000583
RMHTUX02      Q1 PS  00017115/00017094
```

Note: The fact that RMHTUX02 shows up in the queue is purely coincidental. A Linux guest with the on-demand timer is expected to be frequently dropped from queue by the scheduler.

The RMHTUX01 guest seems to stay in the queue all the time. Because the scheduler considers a guest idle after 300 ms of inactivity, this indicates that RMHTUX01 recorded at least 200 timer ticks in a minute interval. Further study reveals the guest consumed some 130 ms of CPU time in that one minute interval. This makes it unlikely the timer is keeping the guest in the queue.

Note: In 7.3.1, “Analyzing the timer ticks” on page 110, we demonstrate how to use the TRACE command to observe timer ticks.

The difference between the two virtual machines is that RMHTUX01 owns a queued direct input/output (QDIO) network device, while RMHTUX02 is connected through IUCV.

After discussion with z/VM Development, we concluded CP did not drop the RMHTUX01 virtual machine from queue because a read I/O event was awaiting

completion on two of its three QDIO devices. A virtual machine normally remains in queue during active I/O (on the expectation that the I/O will complete shortly). In the case of QDIO, the outstanding I/O is not an actual pending read operation. Instead, it is simply part of the protocol. The same situation applies to virtual CTC connections (where an outstanding “read” on one end allows the other end to send data).

APAR VM63282 has been opened to fix this situation in z/VM. We have tested an experimental version of the fix and confirmed that it does indeed allow CP to drop virtual machines from queue. This is confirmed by the output of the CP INDICATE command shown in Example 7-8. Here, 100 Linux guests with the on-demand timer patch applied are connected to a z/VM Guest LAN.

Note: Before applying the fix, these virtual machines all showed up in Q3 and E3.

Example 7-8 Dropping QDIO virtual machines from queue

```
CP IND
AVGPROC-003% 02
XSTORE-000000/SEC MIGRATE-0000/SEC
MDC READS-000001/SEC WRITES-000001/SEC HIT RATIO-090%
STORAGE-052% PAGING-0001/SEC STEAL-000%
Q0-00000(00000)                          DORMANT-00094
Q1-00020(00000)           E1-00000(00000)
Q2-00003(00000) EXPAN-002 E2-00000(00000)
Q3-00014(00000) EXPAN-002 E3-00000(00000)

PROC 0000-004%          PROC 0001-002%

LIMITED-00000
```

Even though these 100 guests wake up every second to perform memory management housekeeping, only 20 on average remain in queue. This allows CP to steal idle pages from those guests as memory becomes constrained.

Note: CP steals idle memory pages from virtual machines in queue. However, it steals pages more aggressively when virtual machines drop from queue.

7.5 Infrastructure cost

Some of the resources used on z/VM to run Linux virtual machines can be seen as infrastructure cost: The resources used to run these utility services are not available for use by Linux virtual machines to run business applications. This

does not mean that these utility services are not necessary, however it does mean that it is often worthwhile to review those services and see whether savings can be realized.

7.5.1 Formatting disks

The zSeries DASD must be formatted for Linux to use it. The dasdfmt program supplied with the Linux distributions can be used for that. Formatting of a 3390-3 volume on RVA takes about 15 minutes elapsed time. As we expected, formatting a disk takes only a very small amount of CPU cycles, about six seconds for a 3390-3 volume.

The real cost for formatting disks comes from the working set of the virtual machine. When you run dasdfmt in the same virtual machine that you just used for compiling the kernel, the entire working set of the virtual machine is kept in storage during the formatting.

Note: Using the formula $Memory \times Time$, we can derive a measure of the relative "cost" of memory usage. Using this calculation, we can express the cost of formatting a 3390-3 volume in a 128 MB Linux guest as 32 MB-hours.

CMS also requires that disks are formatted before use and uses the CMS FORMAT command to do so. Both Linux and CMS normally format the 3390 tracks with 4 K blocks. The difference between Linux and CMS format is minimal. Although using CMS to format the disk uses even less CPU cycles (about 1.6 seconds), the big difference is in the working set of the virtual machine. CMS FORMAT runs with 518 pages, so approximately 0.5 MB-hours for a single volume (1.6% of what Linux uses for the same task).

Note: While it is tempting to view ECKD formatting as the equivalent of a "low-level format" as used on some PC disks, this is not completely correct. With ECKD, you can format and write in a single operation. If you plan to put data on these disks immediately after formatting, you can save a lot of resources by combining these steps. Clearly, you can not use Linux tools to do so, because Linux requires the disk to be formatted separately first.

7.5.2 Installing new systems

There are many different ways to install Linux systems. New systems can be "cloned" from an existing "golden image," or you can do a fresh install for each system. For a fresh install, you can IPL from disk, from tape, or from the virtual card reader. The packages can be loaded from a local disk or from a remote FTP or NFS server. This is true when installing Linux on discrete servers, as well as

when you install Linux in a virtual machine (though with Linux on z/VM you have some extra options to automate the install process).

When installing Linux on discrete machines, the thing that matters most is to get the job done with minimal effort (and in the least amount of time). With Linux on zSeries things are a bit different because the virtual machines share resources. The resources used for the installation of a new Linux system can not be used by other Linux virtual machines running the business applications. Unless your z/VM system has plenty of unused resources, you probably should also concentrate on doing the install using the least amount of resources.

We compared five ways to install a new Linux system on z/VM:

- **RDR + FTP + Router**
  The virtual machine is connected through IUCV to a VM TCP/IP stack as the virtual router. It IPLs from the virtual reader and installs the RPM packages through FTP.
- **RDR + FTP**
  Similar to the first method, but in this case, the Linux virtual machine has its own OSA device. This avoids the cost in the VM TCP/IP virtual router.
- **QuickStart**
  A single (R/O) minidisk is used that holds the starter system and a copy of the RPM packages. The minidisk is IPLed to get the ramdisk system (instead of IPL from virtual reader). The minidisk is then mounted in the ramdisk system to install the packages (which avoids the network traffic and FTP server cost).
- **Breeder**
  A separate Linux virtual machine (the Breeder) is used. The Breeder does a R/W link to the target minidisks of the new system and makes a copy of a preinstalled system onto the target minidisks. After the file system is copied, the “personalization” is done (host name and IP address, for example). This method also avoids the unzipping of the RPM packages.
- **GUI + FTP + Router**
  With SuSE SLES 8, there is a working X-Windows version of YaST. Even though this is a different installer and kernel version, we include it in the measurements to show some of the differences between the install methods.

Note: The QuickStart install method should not be confused with Red Hat Kickstart. Kickstart automates installation by obtaining installation parameters from a configuration file; no user prompting is required. The biggest savings, if any, are in elapsed time for the install, and therefore, also in the memory usage. For SuSE, there are options such as Auto-YaST and Alice, but the versions we have seen still lacked some of the install options that are essential for Linux on zSeries.

For each of the four tests, we installed the same minimal set of packages, just enough to get a working Linux system. The 130 packages in this set are approximately 140 MB worth of RPM files to be loaded. When unpacked during the install, this results in 200 MB worth of data on the root device.

The Breeder install process is a home-grown installation process that we used. Even though the numbers are not immediately applicable to your own installation, we believe it is a representative measurement for what people are doing with various cloning approaches and DDR copies of minidisks.

Elapsed time comparison

In Figure 7-2, we show the elapsed time for the five installation methods. Because each install process uses some manual steps (navigation though YaST screens) this is not easy to measure. To get an idea of the elapsed time, we recorded the CPU usage per minute and discarded the intervals where the Linux system used little more than the idle load.

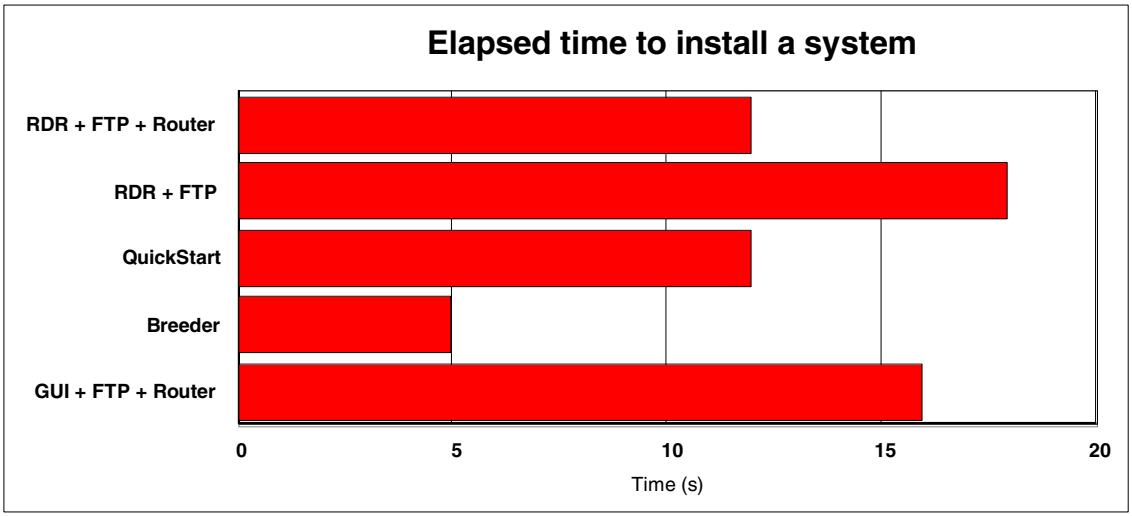


Figure 7-2 Elapsed time for installation using different methods

The Breeder method is clearly the fastest. This should not be a big surprise because it avoids some steps that are known to be relatively slow on zSeries. It is a bit surprising to see that taking out the virtual router does not make the second method faster. The reason for this is that the VM TCP/IP stack is more efficient in driving the OSA device (an OSA-2 Token Ring in this case) than the Linux LCS driver. We believe this difference can be attributed to the use of Diagnose98 in the VM TCP/IP stack.

CPU time comparison

Probably more important than the elapsed time is the CPU time used for the installation. The total CPU cost, as well as the breakdown of the cost per virtual machine, is shown in Figure 7-3.

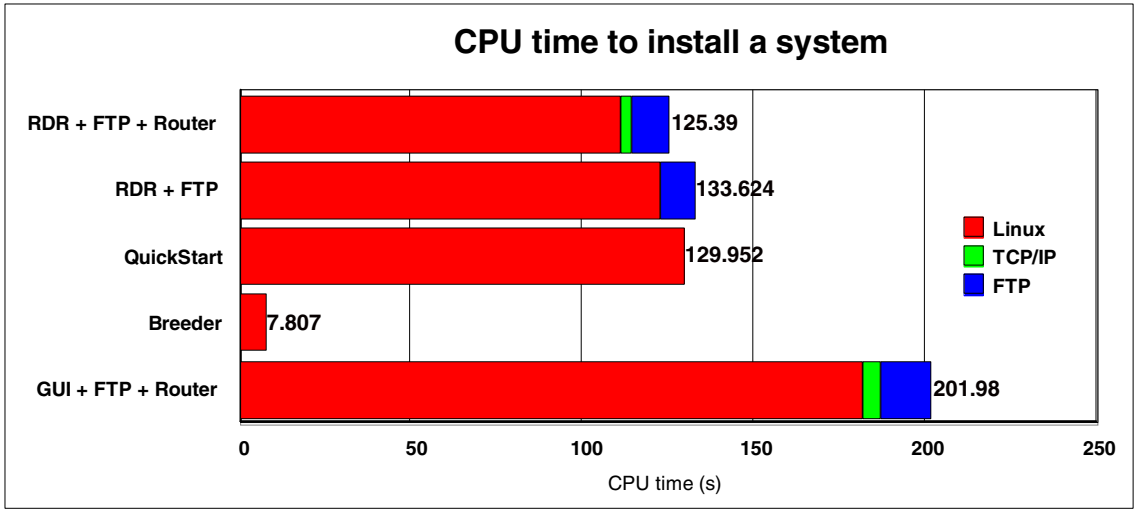


Figure 7-3 CPU time used to install a system

The Breeder method uses less CPU cycles because it avoids the cost decompressing the RPM packages. The differences between the other methods are less obvious. For the installs that use an FTP server, the cost of the FTP server are roughly the same. The main difference between the first two methods must be attributed to the more efficient LCS device driver in VM TCP/IP. The GUI method of installation uses a lot more CPU cycles because of the X-Windows applications and the additional network traffic (the CPU portion for TCP/IP is slightly larger).

If the installation was done on a more CPU-constrained system, the larger demand for CPU cycles would have immediately translated into longer elapsed time as well.

DASD I/O comparison

Another cost item to look at is DASD I/O. Because the QuickStart method copies the RPM packages from disk instead of receiving them through the network, we expect a larger DASD I/O rate (but far less than double because RPM packages are compressed). This is confirmed by the graph in Figure 7-4 on page 119.

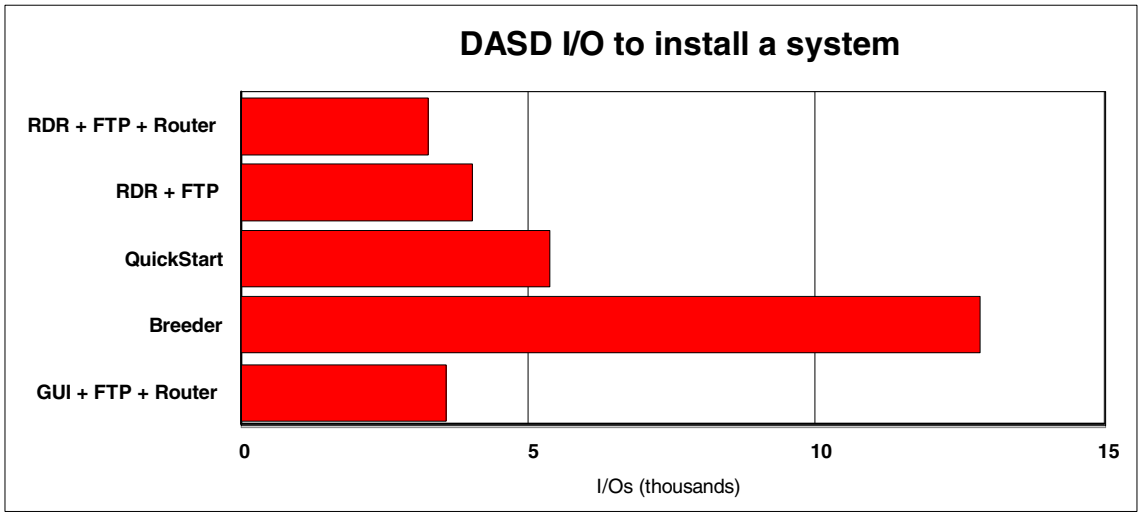


Figure 7-4 DASD I/O to install a system

It is not clear whether the difference between the two FTP methods is significant, but it is not unlikely that a Linux program will do more I/O for the same work when it takes longer to complete (see 8.5.1, “Parameters for bdflush” on page 137).

The increased amount of DASD I/O for the Breeder installation is significant. The Breeder uses an uncompressed copy of the file system to install on the target disks (this saves a lot of CPU time). The drawback of that approach is that we do more I/O while saving CPU cycles. This is a very obvious example of the trade-off you make when tuning a system. When you need to install a lot of systems in a short time, z/VM minidisk cache will do you a lot of good. It might even be attractive to make the Breeder Linux virtual machine big enough to hold the entire image of the target disk in buffer cache.

Note: Because the Breeder first copies the target disk and then applies the personalization, it is possible to build a stock of copied target disks. The only thing left would be personalization after the host name and IP address are known. The stock could be replenished during the night or some other time when sufficient resources are available to do so. It would also be possible to have a CMS service machine combine the formatting with copying the image (see 7.5.1, “Formatting disks” on page 115).

The CLONEDISK support added to DirMaint with APAR VM63122 is also meant to offer some interesting options in this area. Unfortunately, the PTF for this APAR only became available at the end of the writing of this book, so we were not able to experiment with this new code.

Memory usage comparison

The final cost factor for installing Linux systems is memory utilization. While we do not have full numbers on the working set of the virtual machines during the install process, for Linux virtual machines, it is normally good enough to multiply the virtual machine size with the elapsed time. For the ramdisk installation system, the virtual machine size needed is approximately 128 MB.

For the graph in Figure 7-5, we included part of the memory usage of the FTP server and the TCP/IP stack where applicable

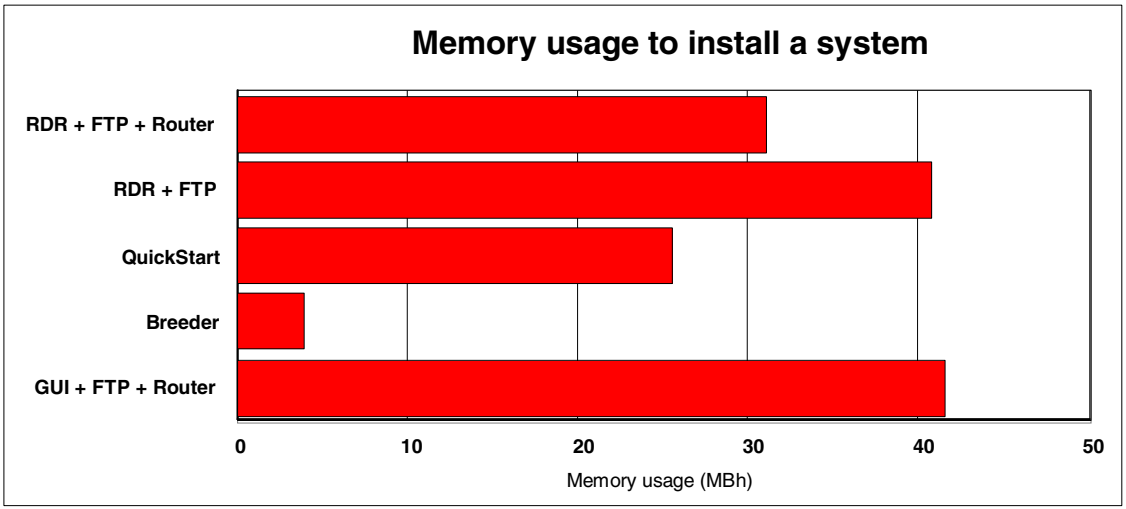


Figure 7-5 Memory usage for installation of a system

Conclusion

Considerable savings can be made by using an efficient installation process. This will be an important issue if you create a lot of Linux systems or when you want to deliver new systems very quickly.

7.6 Performance effect of virtual processors

Assigning virtual processors to a Linux virtual machine can be an effective means of matching zSeries resources to anticipated workload. Overall throughput can be affected by the number processors assigned to a Linux virtual machine.

7.6.1 Assigning virtual processors to a Linux guest

CP shares all real processors defined to an z/VM LPAR. All virtual machines appear to have at least one processor (referred to as the base virtual processor). Additional virtual processors can added to a virtual machine using the CP DEFINE CPU command. As shown in Figure 7-6, virtual processors share real processor resources.

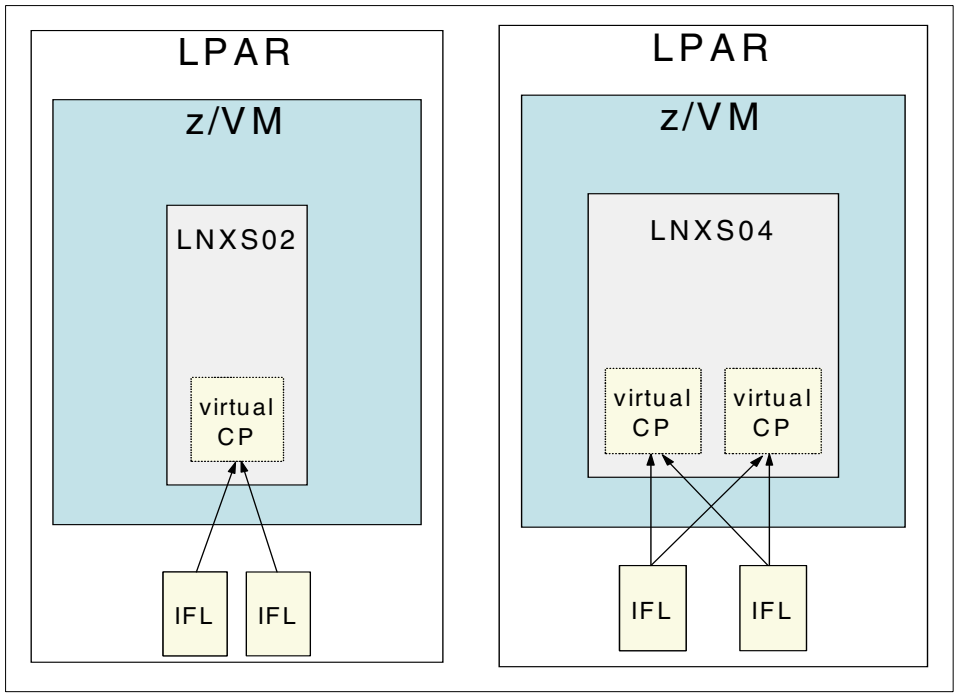


Figure 7-6 Virtual processors in a virtual machine

In the figure, Linux guest LNXS02 has access to two physical IFLs. However, because it is a virtual uniprocessor, only one IFL may run at a time. Linux guest LNXS04 is defined to have two virtual processors; each virtual processor utilizes the physical IFLs in the LPAR. For details about how to configure virtual processors to a z/VM guest, consult *z/VM V4R3.0 Virtual Machine Operation*, SC24-6036.

7.6.2 Measuring the effect of virtual processors

Adding virtual processors to a virtual machine can improve performance for processor-constrained workloads. To examine the effects of defining virtual processors to a Linux guest, we consider the WebSphere Performance Benchmark Sample workload (discussed in Appendix A, “WebSphere

Performance Benchmark Sample workload" on page 155). Using WebSphere Performance Benchmark Sample in a three-tier configuration, we determine the relative cost in terms of processor resources for each component of the workload:

- The WebSphere Application Server
- The HTTP server
- The DB2® server

Note: We run each component in its own virtual machine and measure the processor resources expended during a simulation running five concurrent clients. The processor time spent by each virtual machine is attributed to the WebSphere Performance Benchmark Sample component running in the respective z/VM guest.

In Figure 7-7, we examine the effect varying the number of virtual processors has on CPU utilization. In this scenario, two IFLs are dedicated to the z/VM LPAR. The Linux guests running the IBM HTTP and DB2 servers both run with a single virtual processor (the default). The number of virtual processors allocated to the WebSphere guest is varied from one up to four.

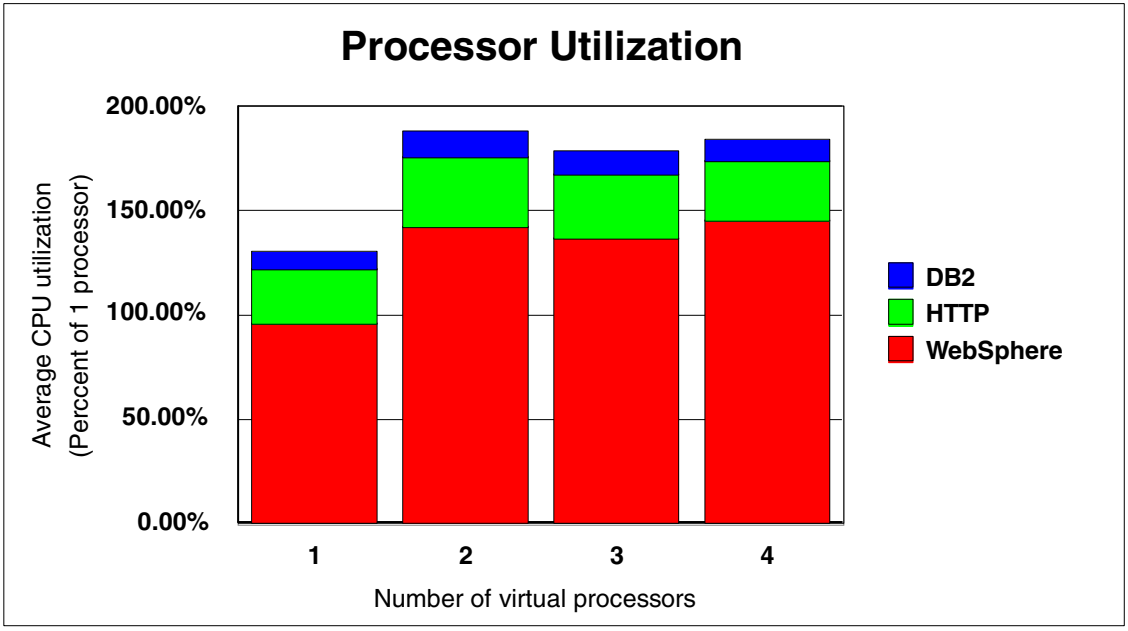


Figure 7-7 Measuring the effect of virtual processors on CPU utilization

From Figure 7-7, we see this workload is processor constrained. With one virtual processor, the guest running WebSphere consumes more than 95% of a single

real processor. When two virtual processors are allocated to the guest, processor utilization increases to 143%. Note that increasing the number of virtual processors beyond the number of real processors does not increase the CPU utilization for the WebSphere guest.

Using the average response time and average CPU time utilization, we derive the cost of adding virtual processors. Using the reported transaction rate and measured average CPU time expended in each Linux guest, we calculate the average cost of a WebSphere Performance Benchmark Sample transaction (measured in milliseconds per transaction) in Figure 7-8.

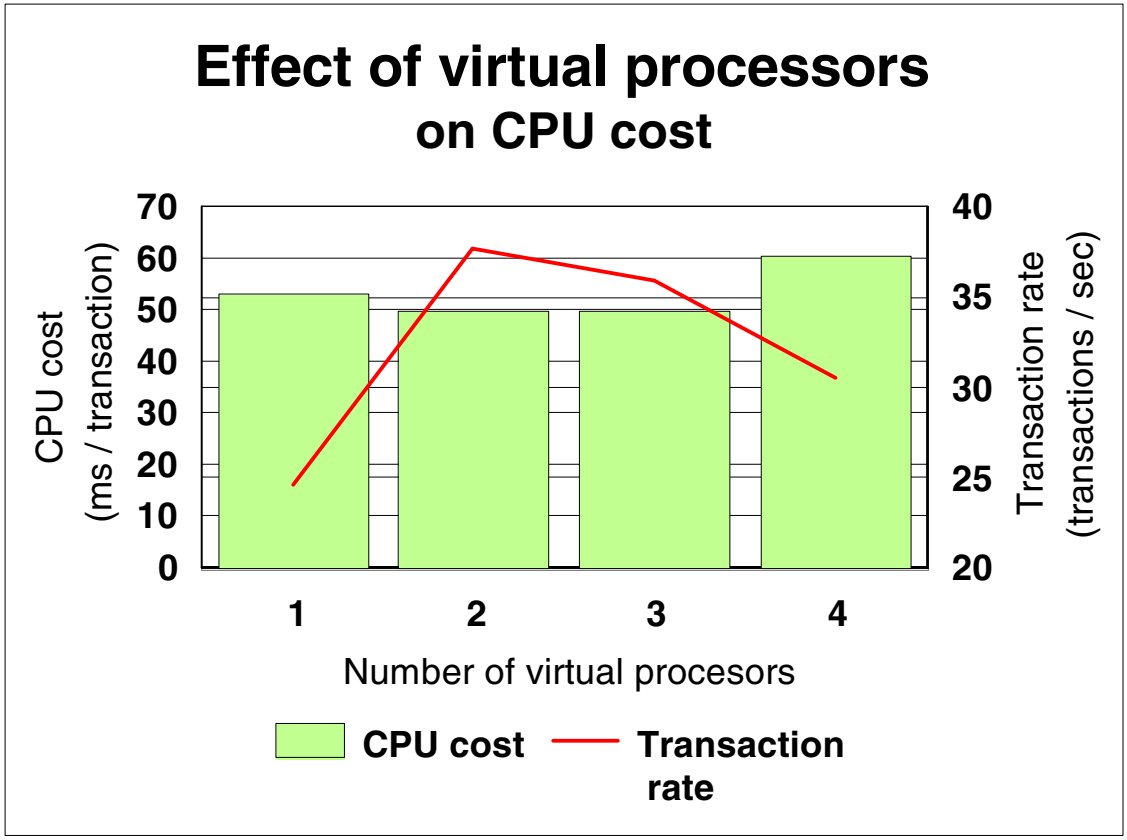


Figure 7-8 Measuring the cost of adding virtual processors

In Figure 7-8, we see that when the number of virtual processors matches the number of real processors, the average transaction cost decreases slightly. More importantly, the average transaction rate increases significantly. Note however, that as the number of virtual processors is increased beyond the number of real processors (two in this case), the overall transaction rate decreases, and the cost

per transaction increases. This is due to the additional scheduling overhead incurred by CP.

When a Linux guest runs a processor-constrained workload, we recommend:

- Defining the same number of virtual processors to the guest virtual machine as the number of real processors available to the LPAR.
- Never defining more virtual processors to the guest virtual machine than the number of real processors available to the LPAR.

8

Tuning DASD performance for z/VM Linux guests

This chapter describes direct access storage device (DASD) tuning for z/VM Linux guests. Topics include:

- Factors that influence DASD I/O
- Using VM DIAGNOSE I/O
- Comparing Diagnose and ECKD I/O
- Comparing ESCON and FICON performance
- Data caching and bdflush

8.1 Factors that influence DASD I/O

When evaluating DASD I/O performance, overall response time consists of several components, as depicted in Figure 8-1.

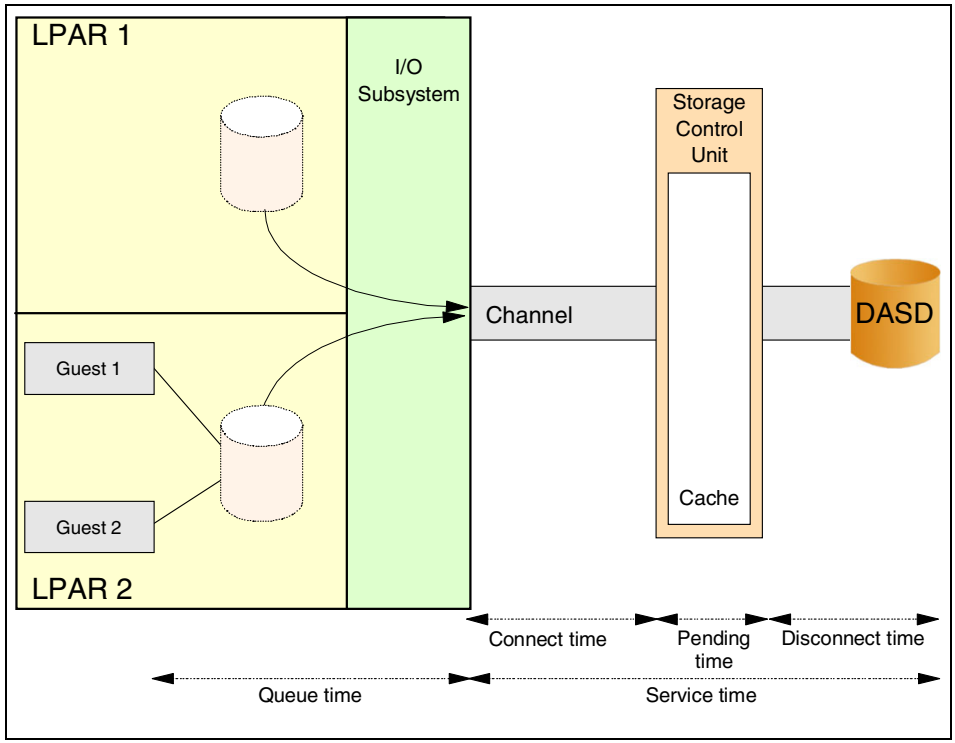


Figure 8-1 Components of overall DASD response time

These components are:

- **Queue time**
  Queue time is a result of multiple users simultaneously accessing a device. If the device is busy servicing an I/O request, additional I/O requests wait in the queue. The length of time an I/O request waits is the queue time. This component is the most variable, and under heavy load, can be the most serious. High queue times can indicate operating system contention, or a high device service time (as I/O requests take longer to complete, requests for service from other users can cause queuing for heavily used devices).
- **Connect time**
  Connect time is the actual time to transfer the data on the channel, normally less than 2 ms for a 4 K block of data.

- **Pending time**
  Pending time is the time required to start an I/O operation. Normally, this amounts to less than 1 ms. High pending time reflects contention on the I/O path. This might be due to contention on the channel, on the control unit, or on the DASD device.
- **Disconnect time**
  Disconnect time is the time required by the control unit to access the data. This includes rotational delays, seek delays, cache management, and processing time inside the control unit. Disconnect time is normally less than 2 to 3 ms on cache controllers.
- **Service time**
  Service time is the sum of pending, connect, and disconnect times.
- **Response time**
  Response time is the sum of queue and service times.

8.1.1 General DASD I/O recommendations

In general, to optimize I/O performance, use more parallelism when possible.

Use more, smaller disks

I/O operations to a single physical disk are serialized. Therefore, when defining the disk layout within the storage controller, smaller unit types (such as a 3390 Model 3) are preferable to the larger 3390 Model 9. The storage capacity of a 3390-9 is equivalent to three 3390-3 units. However, three simultaneous I/O operations are possible using the 3390-3 configuration, while only one is possible for the 3390-9.

Spread data over several units

With Logical Volume Manager (LVM) data striping and software RAID emulation (RAID 0), a single logical disk is mapped to several physical disks. This allows the system to initiate up to as many simultaneous I/O operations as physical disks. However, be aware that the actual number of parallel operations is also limited by the number of channels (CHPIDs) involved. To avoid contention, place each stripe on its own CHPID. When striping, contention can also occur at the control unit, or at the DASD unit. Ensure that the disks are distributed to different control units and define the minidisks on separate DASD units.

Note: Tests performed by the Boeblingen lab indicate optimal stripes are 32 K and 64 K in size.

Figure 8-2 on page 128 illustrates how LVM can increase I/O performance.

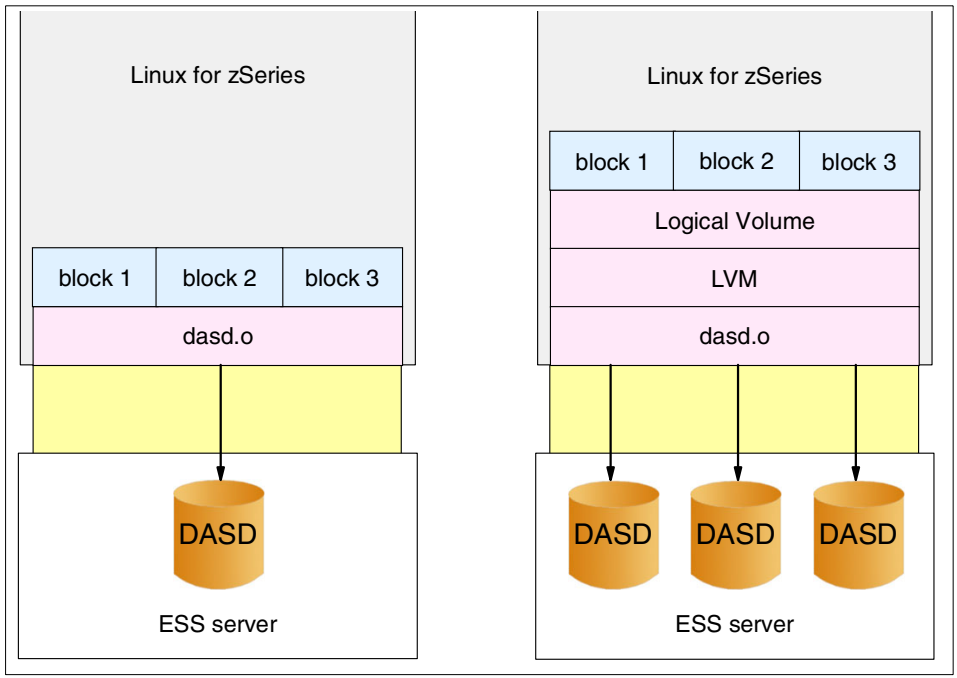


Figure 8-2 Using LVM for parallel DASD access

Using LVM, data is stripped across multiple physical DASD devices. DASD accesses can proceed in parallel. In 8.4.2, “Measuring ESCON and FICON for multiple DASD devices” on page 136, we measure the performance gained by using LVM.

Reduce contention points

Contention for I/O resources can occur at several points in the I/O path:

- **DASD contention**
  Multiple virtual machines can experience DASD contention when simultaneously accessing data on the same DASD device.
- **Control unit contention**
  Contention can occur when two or more DASD devices share a common control unit.
- **Channel contention**
  Contention can occur during simultaneously access of control unit or DASD devices sharing the same channel.

Use the z/VM minidisk cache (MDC)

The minidisk cache can be effective in reducing I/O requirements. There are different options for the MDC configuration that will impact the DASD subsystem differently.

8.2 Using VM DIAGNOSE I/O

DIAGNOSE I/O is a high-level protocol that allows a virtual machine to access blocks on its minidisks with less overhead than pure S/390 channel programs with Start Subchannel (SSCH).

To use Diagnose I/O, a minidisk must be CMS formatted and reserved using the following commands:

```
FORMAT 203 D
RESERVE LINUX DIAG D
```

The 203 minidisk is formatted with a default block size of 4 K and accessed as a D disk. All available space is then reserved to a file called LINUX DIAG (the file name is unimportant).

Tip: If using DirMaint, you can request the disk to be CMS formatted by z/VM in the background, using the DIRM AMD command:

```
DIRM FOR SUSE24 AMD 203 3390 1 1500 LXVOL1 BLKSIZE 4096 LABEL LXDIAG
```

This allocates minidisk 203 for guest SUSE24 on DASD volume LXVOL1. The minidisk is allocated starting at cylinder 1 and sized to be 1500 cylinders. The minidisk is labelled LXDIAG and formatted in 4 K blocks.

Remember to RESERVE the minidisk after issuing the DIRM command.

Recent SuSE kernels are built with the DIAGNOSE module (CONFIG_DASD_DIAG). However, the option to load it automatically whenever a DIAGNOSE device is available (CONFIG_DASD_AUTO_DIAG) is not set. Such a disk will be picked up at IPL time by the ECKD driver, as shown in Example 8-1 for device 203.

Example 8-1 IPL with the default ECKD discipline

```
# cat /proc/dasd/devices
0201(ECKD) at ( 94:  0) is dasda:active at blocksize: 4096, 36000 blocks, 140 MB
0202(ECKD) at ( 94:  4) is dasdb:active at blocksize: 4096, 564840 blocks, 2206 MB
0203(ECKD) at ( 94:  8) is dasdc:active at blocksize: 4096, 14040 blocks, 54 MB
```

To pick-up a disk with the DIAGNOSE discipline, do one of the following:

- Recompile the kernel after having selected the two DIAGNOSE-related options **Support for DIAG access to CMS reserved Disks** and **Automatic activation of DIAG module**.
- Or pick the device manually:

  a. Disable the device using the command:

  ```
  echo "set device range=203 off" >> /proc/dasd/devices
  ```

  b. Load the DIAGNOSE driver using the command:

  ```
  modprobe dasd_diag_mod
  ```

  c. Enable the device back:

  ```
  echo "set device range=203 on" >> /proc/dasd/devices”
  ```

DASD device 203 is accessed with DIAGNOSE I/O, as seen in Example 8-2.

Example 8-2 Accessing the 203 DASD device with DIAGNOSE discipline

```
0201(ECKD) at ( 94:  0) is dasda:active at blocksize: 4096, 36000 blocks, 140 MB
0202(ECKD) at ( 94:  4) is dasdb:active at blocksize: 4096, 564840 blocks, 2206 MB
0203(DIAG) at ( 94:  8) is dasdc:active at blocksize: 4096, 14040 blocks, 54 MB
```

8.3 Comparing Diagnose and ECKD I/O

Bonnie is a well-known open source disk and file system benchmark. It is included in most of the distributions, or can be obtained from the following address:

http://www.textuality.com/bonnie/

Bonnie is usually not very well suited for zSeries platform, because it is single threaded and does a lot of character I/O testing. But we just wanted an easy tool to generate I/Os so that we could compare the behaviors of ECKD and DIAGNOSE disciplines. We used a modified version of Bonnie that allows us to explicitly specify which kind of testing we want to perform.

Using a SuSE SLES8 guest with 128 MB of memory, we ran Bonnie against different disks configurations. The test scenario is as follows:

1. Write a 1500 MB file (block sequential I/O).
2. Read the 1500 MB file (block sequential I/O).
3. Bonnie creates four child processes. Each child then executes 4000 seeks to random locations in the file. On 10% of the seeks, the block is read, modified, and rewritten to the file.

The results for a test against an ECKD device are reported in Example 8-3.

Note: In this analysis (as in all others in this book), DASD devices reside on RAMAC Virtual Array (RVA) units.

Example 8-3 DASD performance analysis for ECKD

```
Screen: ESADSD2  ITSO                              ESAMON V3.3  02/25 12:38-12:44
1 of 3  DASD Performance Analysis - Part 1         USER lnxz02  DEVICE  2064 COECB


           Dev        Device %Dev <SSCH/sec-> <-----Response times (ms)--->
 Time       No. Serial Type   Busy   avg  peak  Resp  Serv  Pend  Disc  Conn
 -------- *--- ------ ------ ---- *---- ----- ----- ----- ----- ----- -----
 12:44:00 670E LX670E 3390-3 55.5  74.0  74.0   7.5   7.5   0.2   0.0   7.2
 12:43:00 670E LX670E 3390-3 97.8  26.3  26.3  37.2  37.2   0.3   0.1  36.9
 12:42:00 670E LX670E 3390-3 96.5  24.4  24.4  39.5  39.5   0.3   0.6  38.6
 12:41:00 670E LX670E 3390-3 99.2  19.6  19.6  50.7  50.7   0.3   2.9  47.5
 12:40:00 670E LX670E 3390-3 99.4  19.5  19.5  50.8  50.8   0.3   2.7  47.8
 12:39:00 670E LX670E 3390-3 55.7  11.1  11.1  50.3  50.3   0.3   2.0  48.1
```

Although the very high response time of 50 ms would be worrying in real life, it is to be expected here. Bonnie is made for stressing the disk.

A more interesting result is the large percentage of response time expended in connect time (the effective time spent transferring data on the channel). This indicates I/Os are not delayed by waits. Rather, each I/O operation is transferring a lot of data, which takes some time. Not much can be done to improve the response time of the disk apart of buying newer, faster disks. We could improve the application response time by spreading the data over several disks.

Then, we ran the same test using DIAGNOSE I/O and compared the behavior in Example 8-4 on page 132.

Example 8-4 DASD performance analysis for DIAGNOSE

```
Screen: ESADSD2  ITSO                              ESAMON V3.3  02/25 13:27-13:33
1 of 3  DASD Performance Analysis - Part 1      USER lnxz02 DEVICE 6 2064 C0ECB


          Dev        Device %Dev <SSCH/sec-> <-----Response times (ms)--->
 Time      No. Serial Type   Busy   avg  peak  Resp  Serv  Pend  Disc  Conn
 -------- *--- ------ ------ ---- *---- ----- ----- ----- ----- ----- -----
 13:33:00 6612 LX6612 3390-3 75.8  95.3  95.3  10.7   7.9   0.2   0.0   7.7
 13:32:00 6612 LX6612 3390-3 89.8  43.7  43.7  31.2  20.5   0.3   0.1  20.2
 13:31:00 6612 LX6612 3390-3 98.3  45.3  45.3  33.9  21.7   0.3   0.1  21.3
 13:30:00 6612 LX6612 3390-3 99.7  38.2  38.2  40.5  26.1   0.3   0.9  24.9
 13:29:00 6612 LX6612 3390-3  101  39.5  39.5  39.5  25.6   0.3   0.4  24.9
 13:28:00 6612 LX6612 3390-3 42.3  16.4  16.4  42.1  25.8   0.3   0.5  25.0
```

From this screen, we notice that:

- **More I/O operations are performed, but the average duration is shorter.** The Start SubChannel (SSCH) column reports the number of I/O operations. The average I/O duration is derived from the service time (Serv) column. This result is beneficial for shared environment. I/O resources are held for less time, and therefore, more available for other guests.
- **DASD response times are better than with ECKD.** Although the measured DIAGNOSE response time is about 10 ms less than ECKD response time, very little can be inferred from the result:
  - The response time is better, because an average DIAGNOSE I/O operation transfers less data than ECKD.
  - More total I/O operations are performed with DIAGNOSE in order to transfer the same amount of data.

Although the majority of service time is due to connect time, it is only part of the total response time: Response time = queue time + service time. Queue time measurements against the DIAGNOSE disk are shown in Example 8-5 on page 133.

Example 8-5 DASD performance analysis for DIAGNOSE queue time

```
Screen: ESADSD2  ITSO                              ESAMON V3.3  02/25 13:27-13:33
2 of 3  DASD Performance Analysis - Part 1         USER lnxz02   DEVICE 2064 COECB


          Dev        Device %Dev <SSCH/sec-> <--Queueing-->
Time      No. Serial Type   Busy   avg  peak DASD Cntl  THR
-------- *--- ------ ------ ---- *---- ----- ---- ---- ----
13:33:00 6612 LX6612 3390-3 75.8  95.3  95.3  2.8  0.0  0.0
13:32:00 6612 LX6612 3390-3 89.8  43.7  43.7 10.7  0.0  0.0
13:31:00 6612 LX6612 3390-3 98.3  45.3  45.3 12.1  0.0  0.0
13:30:00 6612 LX6612 3390-3 99.7  38.2  38.2 14.4  0.0  0.0
13:29:00 6612 LX6612 3390-3  101  39.5  39.5 13.9  0.0  0.0
13:28:00 6612 LX6612 3390-3 42.3  16.4  16.4 16.3  0.0  0.0
```

We observe some contention for the disk in the operating system:

- DIAGNOSE generates more, shorter I/Os.
- The four Bonnie child processes running in parallel compete for access to the same disk.

From this approach, it is difficult to clearly understand which of these methods is best suited for Linux under z/VM. Because Bonnie is stressing the disk, in both cases, we note the device is busy more than 95% of the time in the middle of the run. This leaves no room for other potential users to access the disk.

8.4 Comparing ESCON and FICON performance

Enterprise Systems Connection, ESCON®, and the newer Fibre Connection, FICON™, provide I/O connectivity based on fiber optic technology. Table 8-1 on page 134 compares some of their capabilities.

Table 8-1 Comparing ESCON and FICON capabilities

	ESCON	FICON (native mode)
Maximum data transfer rate	17 MB/s	100 MB/s
Date transfer mode	Half duplex	Full duplex
Maximum distance	3 km[a]	20 km[b]
Data droop	At 9 km	No
CTC function	Separate	Integrated
Concurrent I/O operations	1	Up to 32

a. 43 km using repeater
b. 100 km using repeater

From a performance perspective, the higher data transfer rate and number of concurrent I/O operations distinguish FICON from ESCON.

Note: For a complete comparison of the capabilities of ESCON and FICON, consult *IBM @server zSeries Connectivity Handbook*, SG24-5444, available at:

http://www.ibm.com/redbooks/abstracts/sg245444.html

For details about ESCON, consult *Enterprise Systems Connection (ESCON) Implementation Guide*, SG24-4662, available at:

http://www.ibm.com/redbooks/abstracts/sg244662.html

FICON is discussed in *FICON Native Implementation and Reference Guide*, SG24-6266, available at:

http://www.ibm.com/redbooks/abstracts/sg246266.html

Using the configurations illustrated in Figure 8-2 on page 128, we measure the throughput rate of ESCON and FICON channels. Tests are run using the standard dasd.o device driver accessing a single DASD device and using LVM parallel access to multiple DASD devices.

Measurements are taken using Bonnie (discussed in 8.3, “Comparing Diagnose and ECKD I/O” on page 130) running on a z900 LPAR with an Enterprise Storage Server (ESS) disk attachment. Tests are run against a SuSE SLES7 distribution running in 31-bit mode.

8.4.1 Measuring ESCON and FICON for a single DASD device

In Figure 8-3, we measure the performance of ESCON and FICON when reading and writing to a single DASD device.

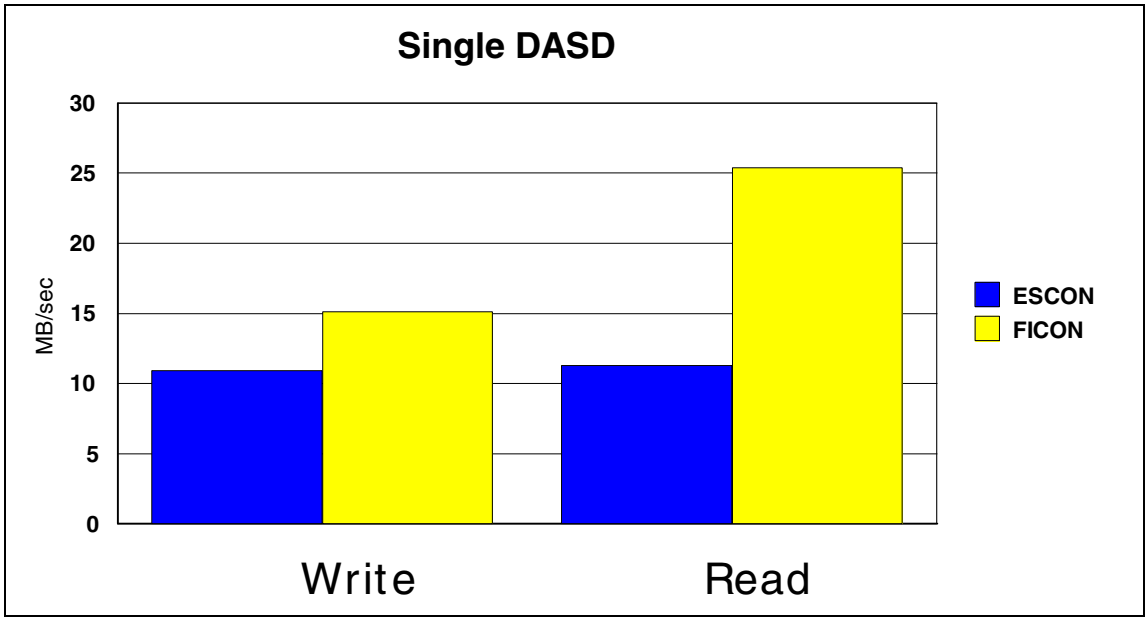


Figure 8-3 ESCON and FICON performance: Single DASD device

Figure 8-3 illustrates a 35% performance improvement for write operations using FICON over ESCON. For read operations, the performance gain is 125%. We also note:

- **The transfer rate for FICON is well below its theoretical maximum.** Although the ESCON transfer rate of 12 MB/s approaches its theoretical limit, the FICON 26 MB/s read transfer rate is significantly less than the 100 MB/s limit. The limitation in this case is not the channel but rather the DASD device itself. To achieve a higher throughput, we need to access multiple DASD devices in parallel.

Note: To maximize FICON performance, use LVM for data stripping to increase parallel DASD access. The performance improvements are shown in 8.4.2, “Measuring ESCON and FICON for multiple DASD devices” on page 136.

- **The FICON transfer rate differs substantially between reads and writes.** To ensure data integrity in the event of a power failure, the ESS server uses nonvolatile storage. This difference is accounted for by the longer write access time to that storage.

8.4.2 Measuring ESCON and FICON for multiple DASD devices

LVM striping enables us to increase the number of parallel I/O operations to DASD devices. In Figure 8-2 on page 128, we examine the effect of parallel access on data transfer rate. In this scenario, Bonnie read scenarios are run using multiple DASD devices combined in a single logical volume. The number of DASD devices is varied from one to eight. Results are compared using eight ESCON and four FICON channels.

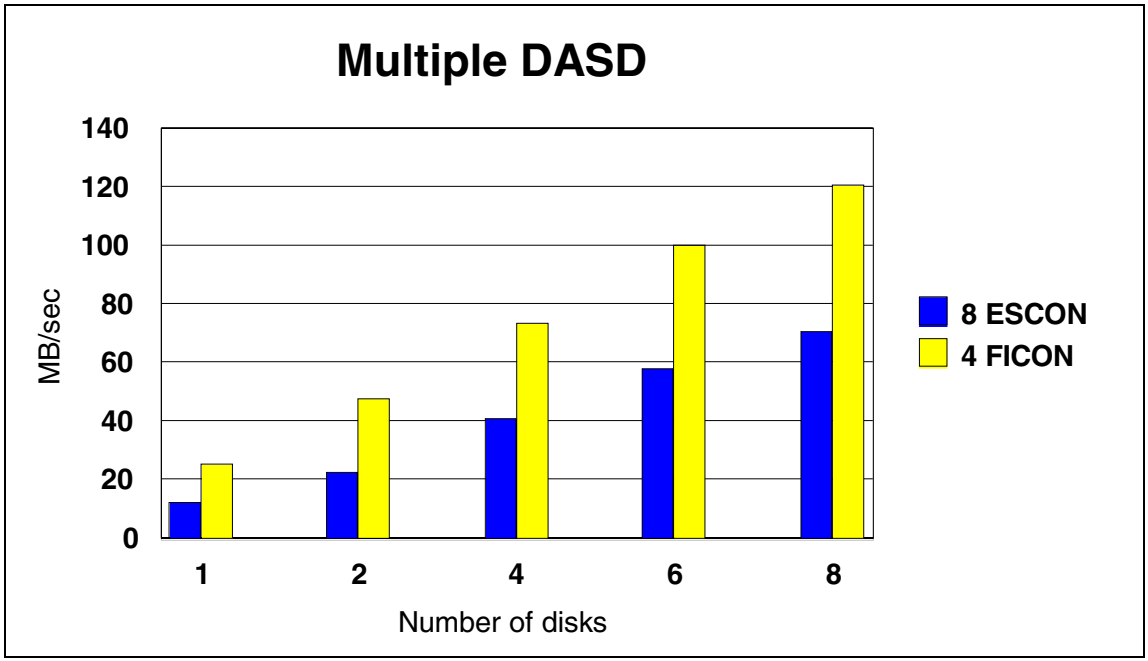


Figure 8-4 ESCON and FICON performance: Multiple DASD devices

Both ESCON and FICON benefit from the increased parallelism provide by LVM. In general, FICON provides twice the data transfer rate (using half the number channels) as ESCON.

8.5 Data caching and bdflush

Having its roots in discrete servers with relatively slow I/O and inexpensive memory, Linux attempts to use memory to avoid disk I/O (see 3.4, “Illustrating Linux aggressive caching” on page 30). One way Linux attempts to avoid I/O is to initially buffer all application output in memory. It then uses an asynchronous process bdflush (or kupdated thread in other releases) to write that data out to disk. This allows for more efficient I/O, helps to keep temporary files away from the disk completely, and allows the operating system to pace I/O in a way that does not impact interactive work on the system.

Note: In this section, we look at the behavior of bdflush. By adjusting its instrumentation, we examine the effect on data caching. However, we can draw no conclusions about what the optimal bdflush parameter settings are for Linux on zSeries.

8.5.1 Parameters for bdflush

To demonstrate the effect of “lazy write,” we run **dt** to write a file slowly (with 64 KB/s) and monitor the disk I/O with **vmstat**.

Note: The Data Test (**dt**) program is a utility to verify operation of peripheral devices. Its syntax and operation is similar to the **dd** command. Details can be found at:

http://www.bit-net.com/~rmiller/dt.html

The results of this experiment are shown in the graph in Figure 8-5 on page 138.

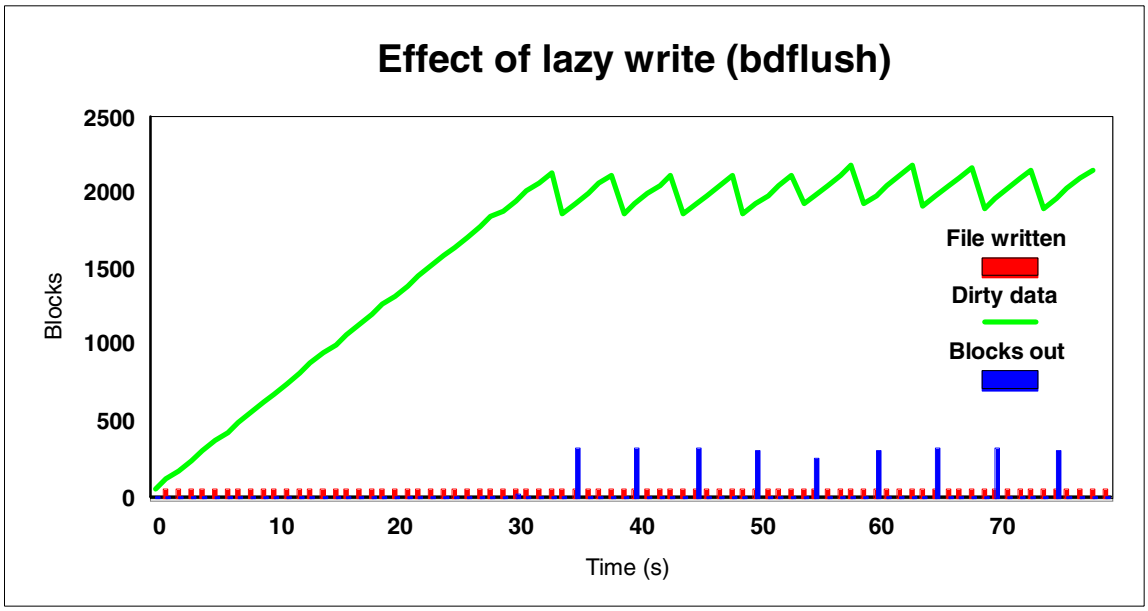


Figure 8-5 Effect of lazy write on slowly writing a file

During the first 35 seconds of the test, there is little to see, except for the small peaks of 64 KB each. Because no data is being written to disk, we must conclude that Linux memory is filling with blocks of data that is to be written to disk (the “dirty data”). After 35 seconds, we see a small amount of data written to disk every five seconds.

The explanation for this strange behavior lies in the default parameters for bdflush. These parameters are controlled through /proc/sys/vm/bdflush. You **cat** the entry to see the current values:

```
# cat /proc/sys/vm/bdflush
30      64      64      256     500     3000    60      0       0
```

The parameters can be changed by writing a line with the appropriate values into the bdflush entry.

Note: Some people suggest the meaning of these parameters can be found in the Documentation directory of your kernel source. When we tried to make sense of the values and the effect of changes, we found the vm.txt file is apparently out-of-date. The real meaning (and defaults) come out of fs/buffer.c instead. Be aware that next kernel releases will change this again.

Table 8-2 Parameters for bdflush (with Linux 2.4.7)

Field	Default	Description
nfract	30	Percentage of buffer dirty required to activate bdflush
ndirty	64	Maximum number of buffers to write out per wake-cycle
nrefill	64	Number of clean buffers to obtain each time refill is called
unused	256	
interval	5 * HZ	Delay (in jiffies) between kupdate flushes
age_buffer	30 * HZ	Time for normal buffer to age before flushing
nfract_sync	60	Percentage of buffer cache dirty required to activate bdflush synchronously

With the description and default values for the parameters, we can understand the graph in Figure 8-5 on page 138:

interval Specifies bdflush is to wake up every five seconds.

age_buffer Specifies that a buffer is to be written to disk only when it is older than 30 seconds.

The first data blocks are written to disk only after 35 seconds. When bdflush wakes up five seconds later, it finds another 320 KB of dirty buffers old enough to write to disk. During the entire test, some 2 MB worth of data is waiting to be written to disk.

For the next experiment, we change `age_buffer` to 10 seconds. The graph in Figure 8-6 on page 140 confirms what we expect:

- Data is first written to disk after 15 seconds.
- Every five seconds afterward, another 320 KB is written to disk.

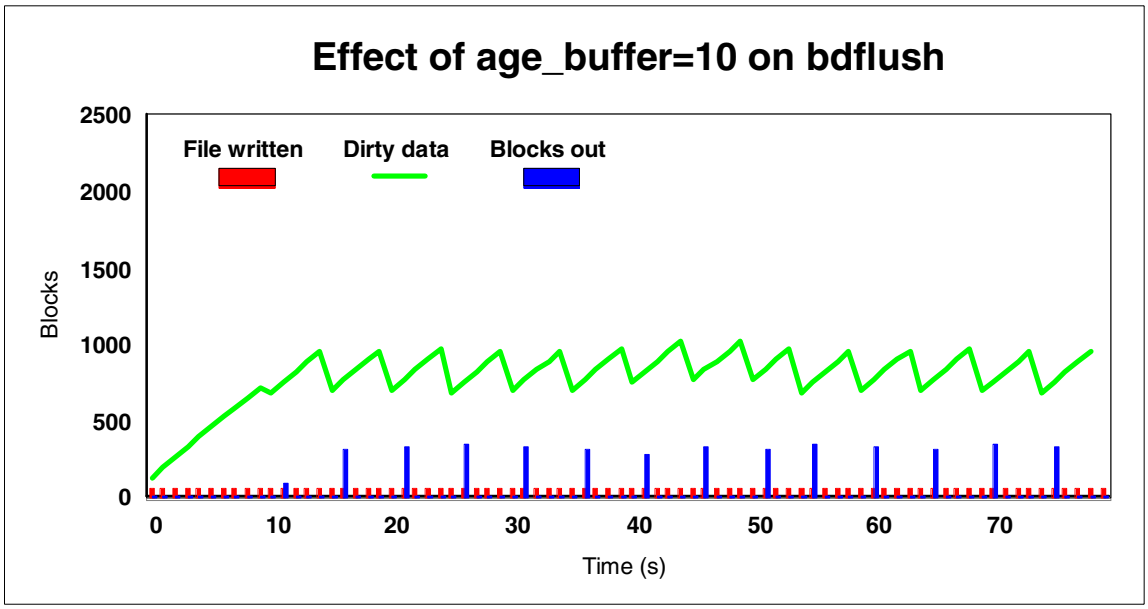


Figure 8-6 Effect of age_buffer=10 on bdflush

It might appear to be a good thing to write the data to disk sooner, but we also find more data is written to disk. We suspect this is due to the same data being written more than once during the test.

Another available tuning method is to increase the interval between flush cycles. When we increase the interval from 5 to 10 seconds, bdflush indeed wakes up every 10 seconds and writes twice the amount of data to disk during each cycle.

Unlike the VM minidisk cache, which uses a write-through cache (as discussed on page 52), Linux utilizes a store-in cache; written data is retained in cache. This is a logical choice, because the data is already buffered in memory before writing it. Much of the cost of the cache is already incurred by the time the data goes to disk.

The effect of buffering only appears when observing the system for a longer period of time. We look at cache growth in Figure 8-7 on page 141.

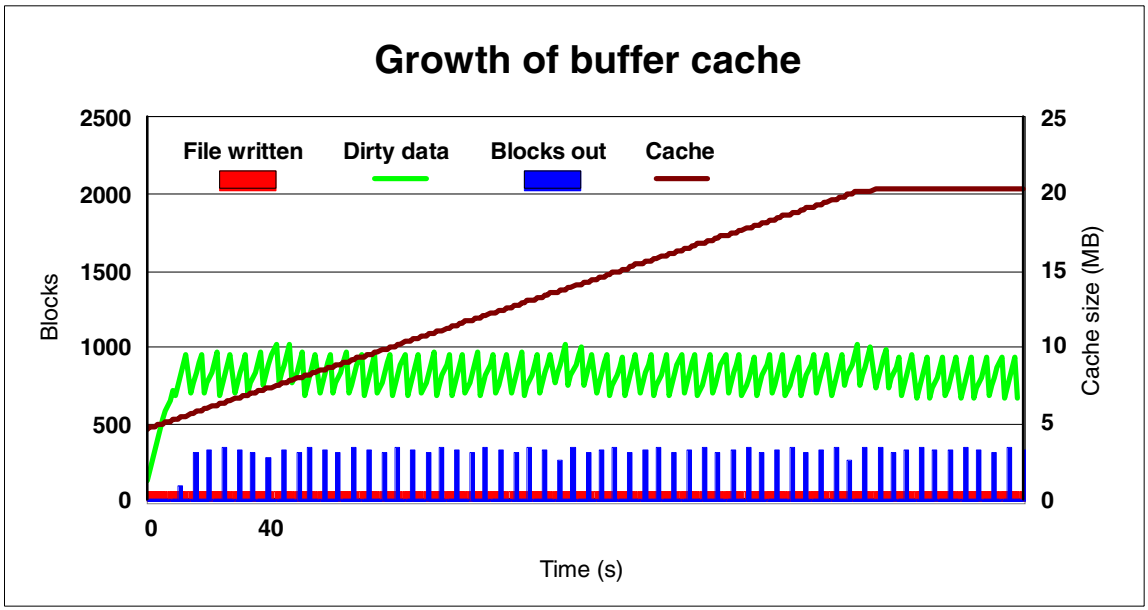


Figure 8-7 Growth of buffer cache while writing a file

The graph shows the cache growing from an initial 5 MB to 20 MB (where it remains). When that point is reached, the oldest blocks are dropped from cache, and memory pages are reused for new data.

The test used here writes a file sequentially, and then reads the file (again sequentially from beginning to end). Due to the both the size of the file and the manner in which the cache is filled when writing the file, the cache offers no benefit when reading the file:

- When reading the file from the beginning, only the last portion of the file remains in the buffer cache from the write phase.
- The portions of the file remaining in the buffer cache from the write phase are evicted to make room for newly read data before the read phase can reuse them.

Note: If the file was small enough to fit in buffer cache, the data could be read back without accessing the disk.

One side effect of increasing `age_buffer` is that the maximum amount of dirty data held in memory changes as well: When bdflush waits longer before writing data, *more* dirty data accumulates in memory. This relationship holds only when data is written at a constant rate. To unconditionally limit the amount of dirty data

in memory, use the `nfract` parameter (the default is to allow 30% of the buffer to be dirty).

Linux on IBM @server zSeries and S/390: ISP/ASP Solutions, SG24-6299 states that the channel programs Linux uses to write out data through bdflush are not always very effective with Linux on z/VM:

- **The dirty pages are left in memory for some time (default 30 seconds) before bdflush starts to write them to disk.**
  This is attractive for temporary files that are discarded before bdflush picks the data up. It completely avoids I/O. However, with Linux on z/VM, it is undesirable to wake up a virtual machine every 30 seconds to write buffer cache data:
  - If there is sufficient contention, z/VM may have already paged these dirty pages.
  - In this was the case, z/VM would first have to page the data back in order for Linux to perform I/O.
- **The channel programs used by bdflush can grow very long, such as 500 KB per I/O.**
  While this is efficient for reducing the number of I/O operations, with Linux on z/VM it can be counterproductive:
  - To start I/O, CP must first lock all Linux memory pages involved in the operation. This can cause contention.
  - Memory pages involved in the I/O operation might have to brought back from paging storage if they were updated long time ago.

9

Measuring the cost of OSA, Linux, and z/VM networking

In this chapter, we look at the relative cost of various networking and routing options when running Linux as a z/VM guest. Topics include:

- Comparing the CPU time cost of routing
- The effect of bandwidth on routing costs
- QDIO optimizations for z/VM
- Memory costs associated with QDIO
- Comparing CPU cost by network type

9.1 Comparing the CPU time cost of routing

To determine the relative cost of various routing options, we use the WebSphere Performance Benchmark Sample workload in the network configurations depicted in Figure 9-1. Details about WebSphere Performance Benchmark Sample can be found in Appendix A, “WebSphere Performance Benchmark Sample workload” on page 155.

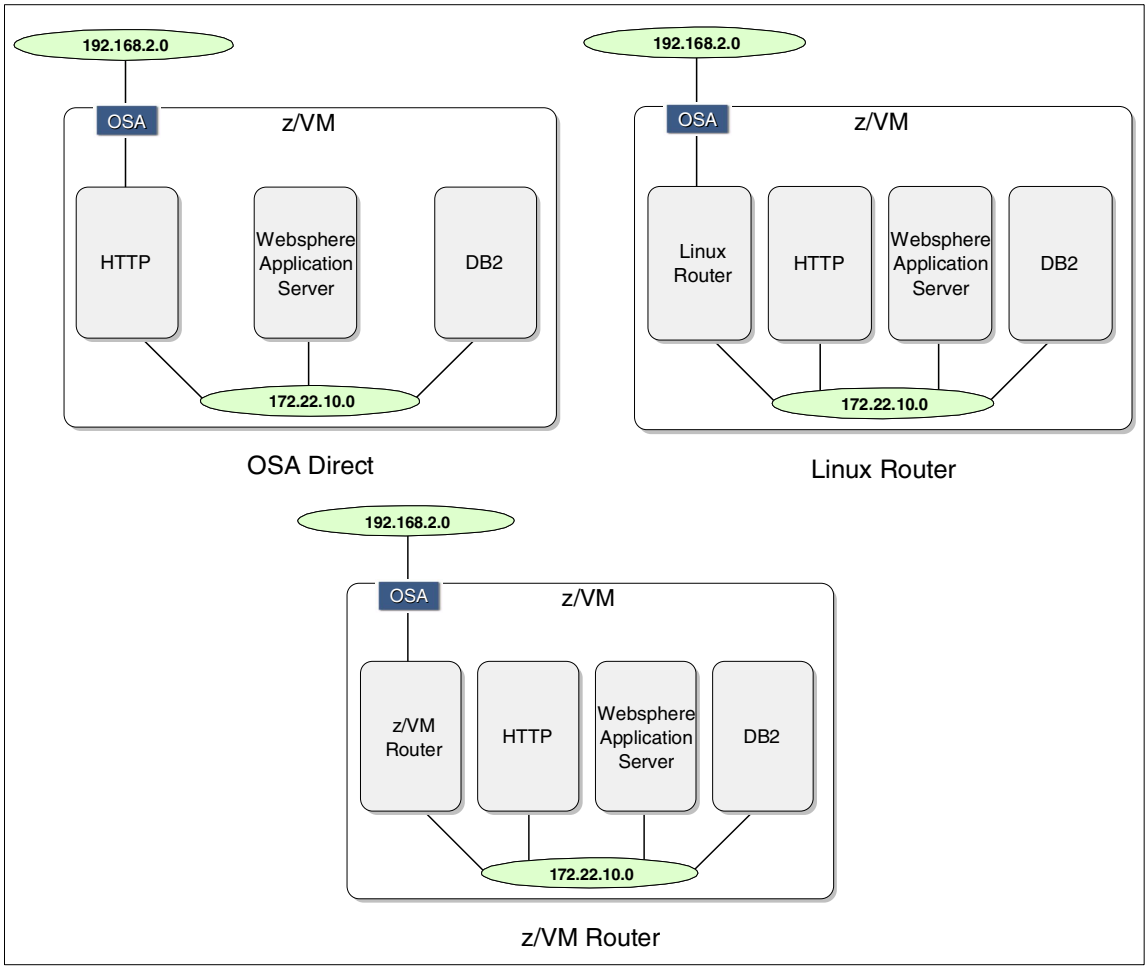


Figure 9-1 Three configurations to measure the cost of routing

We compare the CPU cost for:

- Routing using an Open Systems Adapter (OSA) direct connection
- Routing using a z/VM Linux guest
- Routing using the z/VM TCP/IP stack

In each configuration, we use WebSphere Performance Benchmark Sample in a three-tier configuration. Components are connected using a HiperSockets network (172.22.10.0). Clients connect through an OSA-Express interface connected to a private external network (192.168.2.0).

We establish a common metric to be used when comparing the three routing configurations: the average number of WebSphere Performance Benchmark Sample transactions completed in one CPU second. To calculate this value, we initiate a workload using five simulated clients on the 192.168.2.0 network. We note the reported average transaction rate seen by the clients, as well as the CPU time used by the router guest and the HTTP client. Example 9-1 shows the CPU time consumed by the HTTP server for the OSA direct routing configuration.

Example 9-1 CPU time used by the HTTP server (OSA direct routing)

```
Screen: ESAUSR2  ITSO                                ESAMON V3.3  02/19 12:51-12:56
 1 of 3  User Resource Utilization                    USER lnxs02          2064
C0ECB

                  <---CPU time--> <--------Main Storage (pages)--------->
          UserID  <(seconds)> T:V <Resident>  Lock <-----WSS----->
 Time     /Class  Total  Virt Rat Total Activ  -ed Total Actv  Avg Resrvd
 -------- ------- ----- ----- --- ----- ----- ---- ----- ---- ---- ------
 12:56:00 LNXS02  19.63 11.42 1.7  9807  9807 2482  6981 6981 6981      0
 12:55:00 LNXS02  18.21  9.80 1.9  9807  9807 2574  6981 6981 6981      0
 12:54:00 LNXS02  18.07  9.70 1.9  9804  9804 3047  6981 6981 6981      0
 12:53:00 LNXS02  18.10  9.62 1.9  9804  9804 2974  6981 6981 6981      0
 12:52:00 LNXS02  18.00  9.62 1.9  9804  9804 2401  6628 6628 6628      0
```

Average CPU time used over the interval is 18.4 CPU-seconds/minute. The reported transaction rate over the period is 35.4 transactions/second.

Note: To estimate routing cost, we use the average CPU time (in milliseconds) required for a transaction. In this case, we define a transaction to be an HTTP request and response from the WebSphere Performance Benchmark Sample server. Using the reported average transaction and the CPU cost for each WebSphere Performance Benchmark Sample component, we calculate routing cost from the average CPU time measured for a one minute interval and reported transaction rate.

Results for each routing configurations are summarized in Table 9-1.

Table 9-1 Summary of CPU cost for routing (OSA direct, Linux, z/VM)

Router	Transaction rate[a]	Router CPU time[b]	HTTP CPU time[b]	Total CPU time[b]	Cost[c]
OSA direct	35.4	[d]	18.4	18.4	**8.7**
Linux router	39.2	12.0	13.3	25.3	**10.8**
z/VM router	25.9	4.0	10.2	14.2	**9.2**

a. Transactions/second.
b. CPU-second/minute.
c. CPU-ms/transaction.
d. The OSA direct configuration incurs no CPU cost for a guest router.

The results reveal:

- **The Linux router offers the highest transaction rate.**
  The reported transaction rate for the Linux router is approximately 50% higher than the z/VM router.
- **Cost for an OSA direct connection is least expensive at 8.7 CPU-ms/transaction.**
  This result is expected. Although the CPU time spent by the HTTP server is greater, no additional CPU time is consumed for a router virtual machine.
- **On average, the cost of a Linux router is approximately the same as the cost of an HTTP server when using a z/VM router.**
  Although there are may be good reasons for using a Linux router (such as firewall capability and network packet filtering), there is a measurable cost.
- **The cost of a z/VM router is about 15% less than the cost of a Linux router.**
  The figures indicate z/VM might have some QDIO network optimization not available to Linux guests. We examine this in 9.3, “QDIO optimizations for z/VM” on page 150.

9.2 The effect of bandwidth on routing costs

Using the Iperf tool, we examine bandwidth effects on the routing.

Note: Iperf is a utility for measuring TCP and UDP bandwidth performance. Details can be found at:

`http://dast.nlanr.net/Projects/Iperf/`

Although Iperf supports both TCP and UDP traffic, be aware that the Iperf measurements made in this redbook utilize UDP datagrams. Using the Linux Traffic Control program, tc, we were unable to limit TCP bandwidth only up to 5 Mbit/s. When attempting to limit TCP bandwidth to 10 Mbit/s, Iperf push network load to 94 Mbit/s. However using UDP, Iperf is able to limit network traffic over the full range of bandwidths.

We compare routing using a Linux guest to a z/VM, as shown in Figure 9-2.

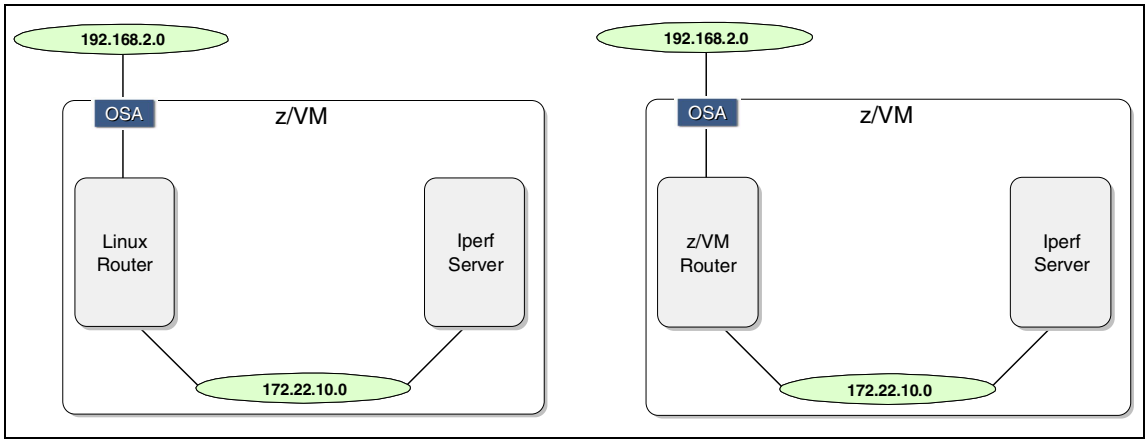


Figure 9-2 Measuring the bandwidth effect on routing cost using Iperf

In the configurations, the Iperf server runs on a Linux guest connected to the router over an internal network (172.22.10.0). The Iperf client runs on a private Ethernet network (192.168.2.0) and access the router through an OSA-Express interface. Each router runs in a 64 MB virtual machine with one defined virtual CPU.

To calculate the effect of bandwidth on routing costs, we use Iperf to drive a network load and measure the CPU time consumed by the router. We use the average data transfer rate reported by Iperf and the measured CPU consumption to derive cost in terms of MB per CPU-second.

Using **Iperf**, network loads can be varied by both consumed network bandwidth and by packet size. To study the effects, we run scenarios that:

- **Vary consumed network bandwidth**
  Scenarios are run with network bandwidth limited to 1 Mbit/s, 25 Mbit/s, 50 Mbit/s, 75 Mbit/s, and 100 Mbit/s using a 600 byte UDP packet size.
- **Vary UDP packet size**
  The UDP packet size is increased to 1470 bytes, and the scenarios are run again at the selected network bandwidths.

- **Vary network type**
  We compare the CPU cost for three types of zSeries networks:
  - HiperSockets
  - Inter-User Communications Vehicle (IUCV)
  - z/VM Guest LAN

Router CPU costs are calculated by dividing the total CPU-ms used by the total data transferred. Results are reported both for virtual CPU time and for CPU time spent in CP.

Note: Virtual CPU time is the actual amount of processing time expended by the z/VM guest. CP time is the amount processor time used by CP acting on behalf of the z/VM guest.

Figure 9-3 on page 149 compares the cost of a Linux router to a z/VM router for varying bandwidths operating on 600 byte UDP packets. CPU costs are broken down for virtual and CP time.

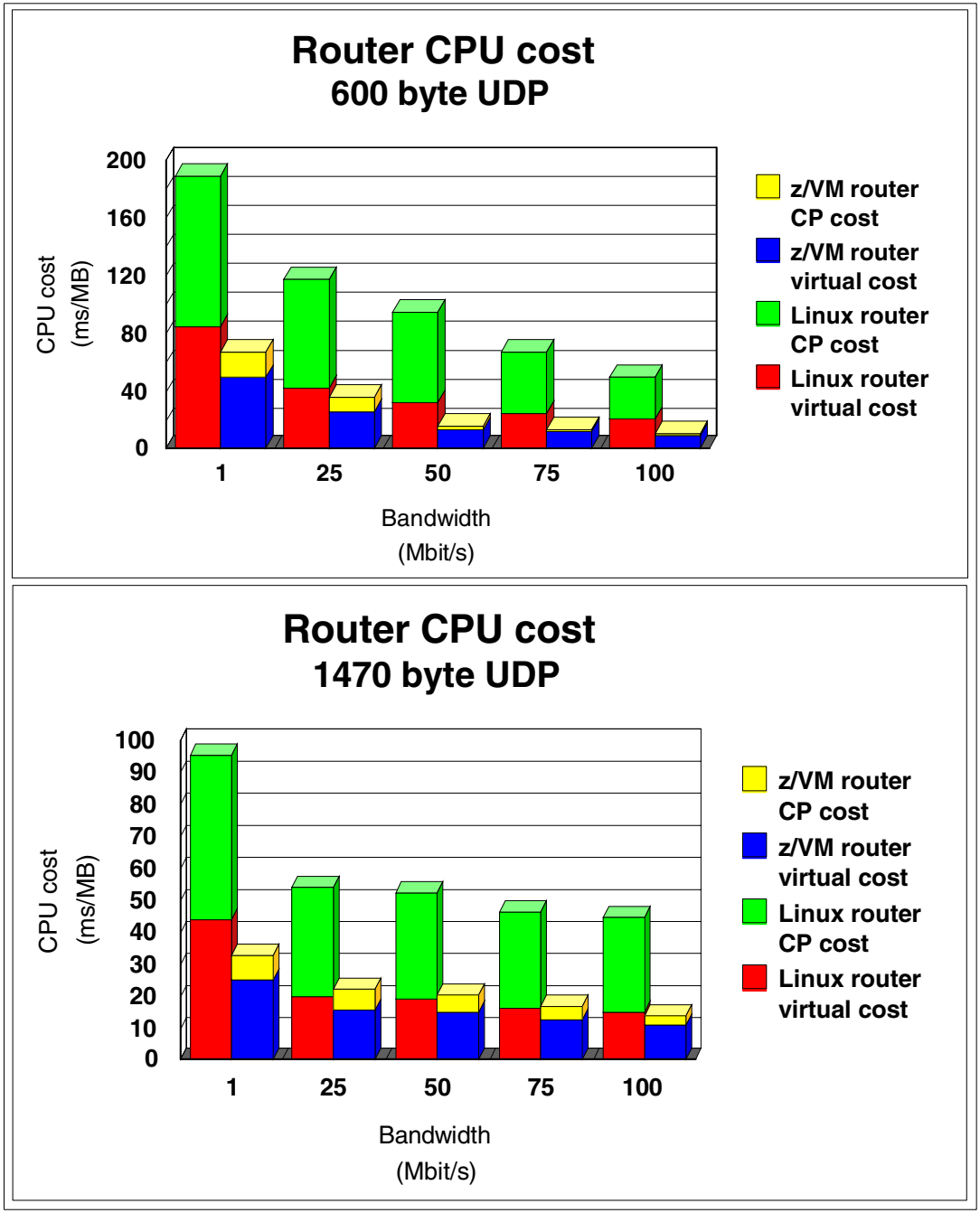


Figure 9-3 Comparing bandwidth effect on z/VM and Linux routing

Note: Take note of the change in y-axis scale between the two charts.

Figure 9-3 on page 149 illustrates that the CPU cost of a z/VM router is significantly less than a Linux guest router, independent of both bandwidth and UDP packet size. A substantial portion of the additional cost of the Linux router is the CPU time spent in CP processing.

9.3 QDIO optimizations for z/VM

The data presented in Figure 9-3 on page 149 is in accord with the conclusion that a z/VM router is more cost effective than a Linux guest router (at least when using OSA-Express interface). We note the routing costs decrease as bandwidth increases, independent of data packet size.

Interestingly, the Linux router spends more than half of its processing time in CP. This can be accounted for by z/VM optimization employed when using QDIO:

- **Use of DIAGNOSE 98**
  z/VM uses the VM DIAGNOSE 98 system service to manage storage for Queued Direct Input Output (QDIO) buffers. This service allows a virtual machine to lock and unlock memory pages. Therefore, the z/VM TCP/IP virtual machine avoids VM hypervisor overhead (and therefore CP processing time) by communicating directly with the QDIO interface.
- **Packing datagrams in the QDIO buffer**
  z/VM has prior knowledge of pending packets destined for QDIO interfaces. The z/VM TCP/IP utilizes this to pack many datagrams into a single network payload (and thereby save CPU cycles when delivering the packed payload to the interface).

  Note: Traffic packing can occur with packets destined for all IP destinations when using an OSA interface. However, only traffic destined to a single IP address can be packaged when using HiperSockets.

These optimizations are available to z/VM on any QDIO interface, OSA or HiperSockets.

Traffic packaging on a QDIO interface can be observed. In Example 9-2 on page 151, we use the **`ifconfig`** command to examine the number of packets received at the Iperf server's HiperSockets interface.

Example 9-2 Counting packets received at the Iperf server

```
hsi1      Link encap:Ethernet  HWaddr 00:00:00:00:00:00
          inet addr:172.22.10.10  Mask:255.255.255.0
          inet6 addr: fe80::200:ff:fe00:0/10 Scope:Link
          UP RUNNING NOARP MULTICAST  MTU:16384  Metric:1
          RX packets:3296073 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1799129 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:1267423148 (1208.7 Mb)  TX bytes:151174713 (144.1 Mb)
          Interrupt:3
```

The `RX packets` value reports the number of received packets. In Example 9-3, we use the CP QUERY VIRTUAL OSA command to count the number of packets received by the router from the OSA-Express interface.

Example 9-3 Counting packets received on the OSA-Express interface

```
OSA  F002 ON OSA   7345 SUBCHANNEL = 0005
     F002 QDIO ACTIVE
     F002 INP + 01 IOCNT = 03160430  ADP = 106 PROG = 000 UNAVAIL = 022
     F002 OUT + 01 IOCNT = 01799105  ADP = 000 PROG = 128 UNAVAIL = 000
     F002 OUT + 02 IOCNT = 00000000  ADP = 000 PROG = 000 UNAVAIL = 128
     F002 OUT + 03 IOCNT = 00000000  ADP = 000 PROG = 000 UNAVAIL = 128
     F002 OUT - 04 IOCNT = 00000023  ADP = 001 PROG = 000 UNAVAIL = 127
```

The `INP IOCNT` value reports the number of received packets.

Tip: z/VM help for the CP QUERY VIRTUAL OSA command says:

```
IOCNT = nnnnnnnn
     specifies the number of data transfers that have completed since the
     QDIO data queue was last activated by the program. CP increments the
     IOCNT each time a QDIO data buffer changes from an adapter-owned
     state to a program-owned state.
```

The QDIO packing effect can be measured by calculating the ratio of packets received on the OSA-Express interface to the number of packets received on the Iperf server HiperSockets interface. In Figure 9-4 on page 152, we show how the packing factor varies by bandwidth using a 600 byte UDP payload.

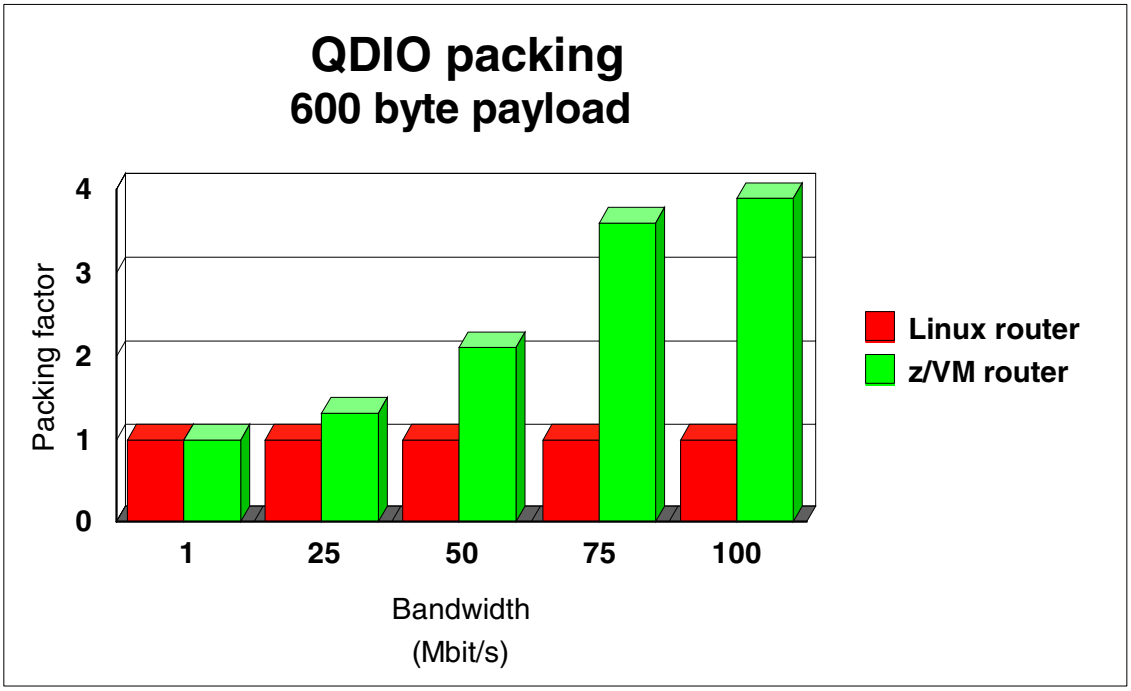


Figure 9-4 Measuring the QDIO packet packing effect

We see the Linux router performs no packing. Its packing factor remains one for all bandwidths. Using a z/VM router, the QDIO packing increases to almost four.

9.4 Memory costs associated with QDIO

Be aware, by default, the Linux QDIO device driver reserves approximately 8 MB of memory *for each QDIO device*, memory that needs to be locked in real memory below the 2 GB address line. This cost should taken into account when running many Linux guest (particularly if each guest uses multiple QDIO devices). For example, the requirement for 50 Linux guests sharing an OSA adapter translates to 400 MB of z/VM memory.

It is possible to change the QDIO device driver memory size. For details, see *Linux for zSeries and S/390 Device Drivers and Installation Commands*, LNUX-1303. Lowering the memory requirement will lower the MTU for the device.

Hint: To conserve memory below 2 GB, consider using a single router (Linux or z/VM) connected to the OSA adapter. Use virtual channel-to-channel (CTC) or IUCV connections from the router to the Linux guests. These devices have much lower memory requirements.

9.5 Comparing CPU cost by network type

In Figure 9-5, we calculate cost of routing for various network types:

- HiperSockets
- IUCV
- z/VM Guest LAN

The comparison is based on Iperf measurements performed on the configuration illustrated in Figure 9-2 on page 147. In this case, a Linux router is used, and the 172.22.10.0 network is varied according to the type under consideration.

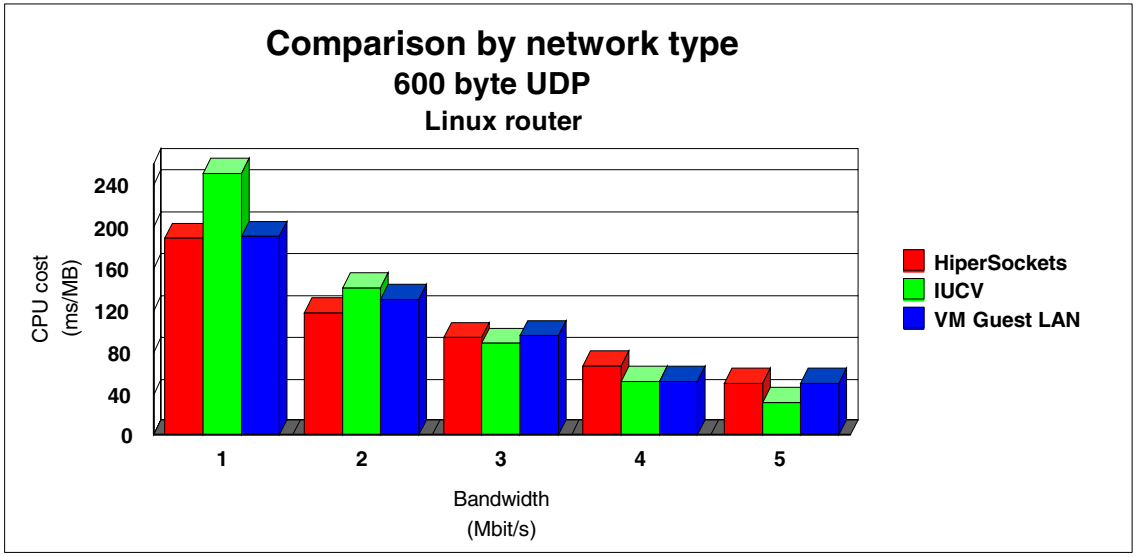


Figure 9-5 CPU cost comparison by network type

The graph illustrates that while the CPU cost of IUCV is slightly higher than HiperSockets and z/VM Guest LAN at low bandwidth, IUCV becomes less expensive at higher bandwidths. We also note the CPU cost of z/VM Guest LAN is essentially equal to the cost of HiperSockets.

A

WebSphere Performance Benchmark Sample workload

This appendix describes the WebSphere Performance Benchmark Sample (WPBS) workload.

WebSphere Performance Benchmark Sample

WebSphere Performance Benchmark Sample (WPBS), commonly referred to as the Trade 2 application, is a sample J2EE benchmark application for IBM WebSphere Application Server. The application simulates a stock trading application. WebSphere Performance Benchmark Sample is available to download for free at:

http://www-3.ibm.com/software/webservers/appserv/wpbs_download.html

WebSphere Performance Benchmark Sample provides a suite of workloads for characterizing performance of the IBM WebSphere Application Server. In this redbook, we use the WebSphere Performance Benchmark Sample to generate workloads that are analyzed in terms of their impact on system performance. Figure A-1 illustrates a logical view of the WebSphere Performance Benchmark Sample application.

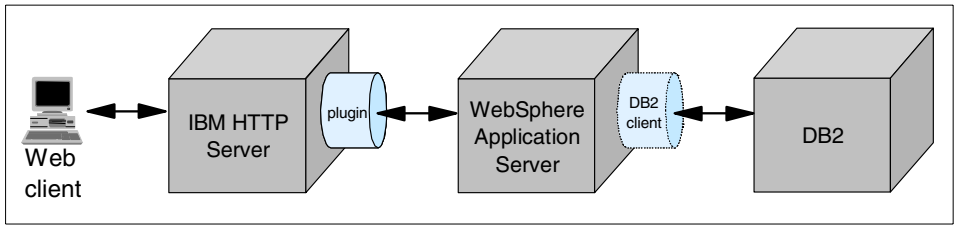


Figure A-1 Components of an IBM WebSphere Application Server deployment

The three main WebSphere Performance Benchmark Sample components are:

- **The IBM HTTP server**
  The HTTP server accepts client requests and delivers static content (HTML pages, images, and stylesheets). Dynamic requests are forwarded to the WebSphere Application Server through a server plugin.
- **The IBM WebSphere Application Server**
  The WebSphere Application Server creates dynamic content using JavaServer Pages (JSP) and Java Servlets. Pages are generated from data extracted from a DB2 database.
- **The DB2 database**
  The DB2 database contains relational tables regarding simulated customer accounts and stock transactions.

WebSphere Performance Benchmark Sample deployment options

Several options are available when deploying the WebSphere Performance Benchmark Sample application on Linux for zSeries:

- All components can run in a single Linux guest.
  We refer to this as a *single-tier* deployment.
- Each component can run in a dedicated Linux guest.
  We refer to this as a *three-tier* deployment.

These two deployment options are depicted in Figure A-2.

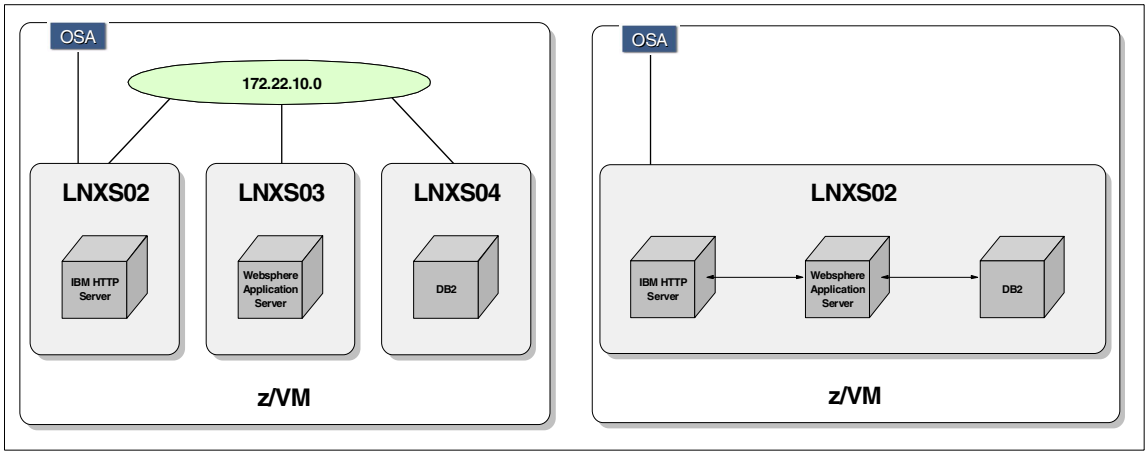


Figure A-2 WPBS deployment: Three-tier versus single-tier

Note: For a complete description of deployment options for WebSphere Application Server, consult *WebSphere Application Server V4 for Linux, Implementation and Deployment Guide*, REDP0405, available at:

`http://www.ibm.com/redbooks/abstracts/redp0405.html`

In this book, the WebSphere Performance Benchmark Sample is used as a workload generator for Linux running as a z/VM guest. In this case, we primarily use the three-tier deployment option. This enables us to adjust the virtual machine size of each Linux guest more accurately based on the task that guest performs. In addition, when measuring utilization, the three-tier deployment option allows us to attribute specific resources usage to a specific task.

B

Mstone workload generator

This appendix describes the Mstone workload generator. Topics discussed include:

- Mstone overview
- Operation of the Mstone workload
- Configuring the Mstone client
- Configuring the example.com domain
- Populating the LDAP database

Mstone overview

Mstone is a performance measurement tool available from the Mozilla project:

http://www.mozilla.org/projects/mstone

Originally known as Mailstone, Mstone can be used for capacity planning and testing network mail servers. The Mailstone user guide can be found at:

http://docs.sun.com/source/816-6036-10/index.html

Using Mstone, workloads can be generated on mail servers from multiple client machines. Test scenarios that employ multiple mail protocols and multiple simulated mail clients can be used to observe server response under heavy load. Although originally intended to test mail server performance, Mstone has features that make it suitable for overall system performance analysis. The Mstone configuration used in this book is shown in Figure B-1.

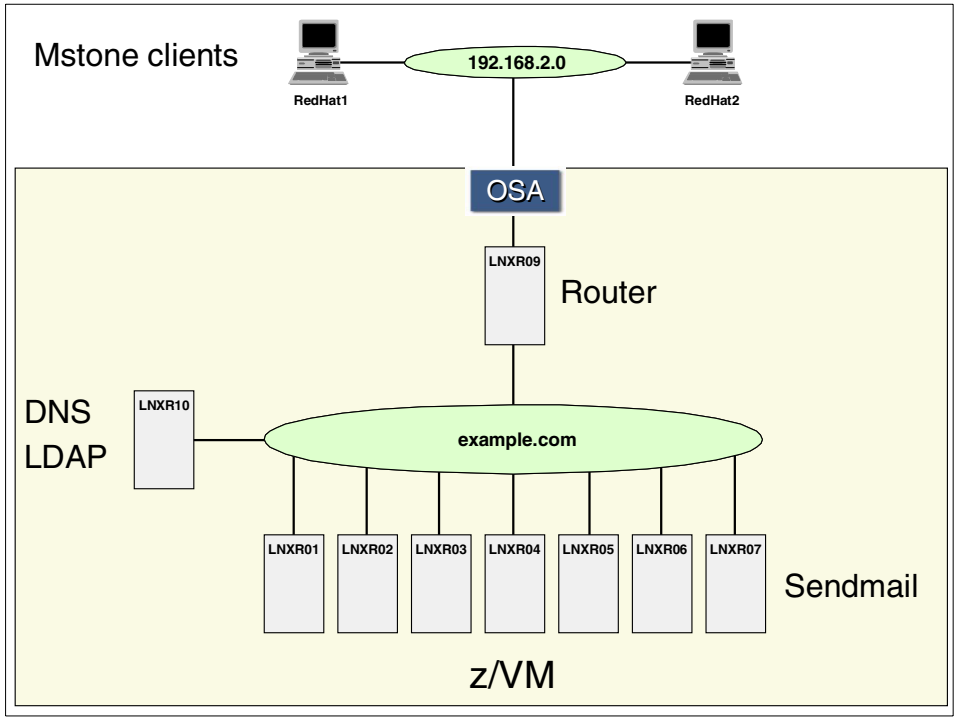


Figure B-1 Mstone workload configuration

The configuration consists of:

- **Mstone clients**
  Mstone clients (hosts RedHat1 and RedHat2) are used to generate mail messages destined to users in the example.com domain. The RedHat1 and RedHat2 Linux hosts reside on a private network routed to the example.com domain through an OSA-Express interface.
- **Linux router**
  The example.com network is implemented as a HiperSockets network. Routing to and from the 192.168.20 private network is managed by a z/VM Linux guest (host LNXR09) configured with IP forwarding enabled.
- **Combined DNS and LDAP server**
  Host name resolution on the example.com network is managed by a DNS server running on the LNXR10 z/VM Linux guest. The OpenLDAP server resolves where specific user accounts reside in the example.com domain.
- **Sendmail servers**
  Messages are routed to the intended recipients in the example.com domain by the sendmail server running on hosts LNXR01 through LNXR07.

Each sendmail host in the example.com domain services 1000 users. The distribution of users across the sendmail hosts is shown in Table B-1.

Table B-1 Distribution of users to sendmail hosts

Sendmail host name	User distribution
LNXR01	itso-user0 through itso-user999
LNXR02	itso-user1000 through itso-user1999
LNXR03	itso-user2000 through itso-user2999
LNXR04	itso-user3000 through itso-user3999
LNXR05	itso-user4000 through itso-user4999
LNXR06	itso-user5000 through itso-user5999
LNXR07	itso-user6000 through itso-user6999

Operation of the Mstone workload

The flow of the Mstone workload is illustrated in Figure B-2 on page 162.

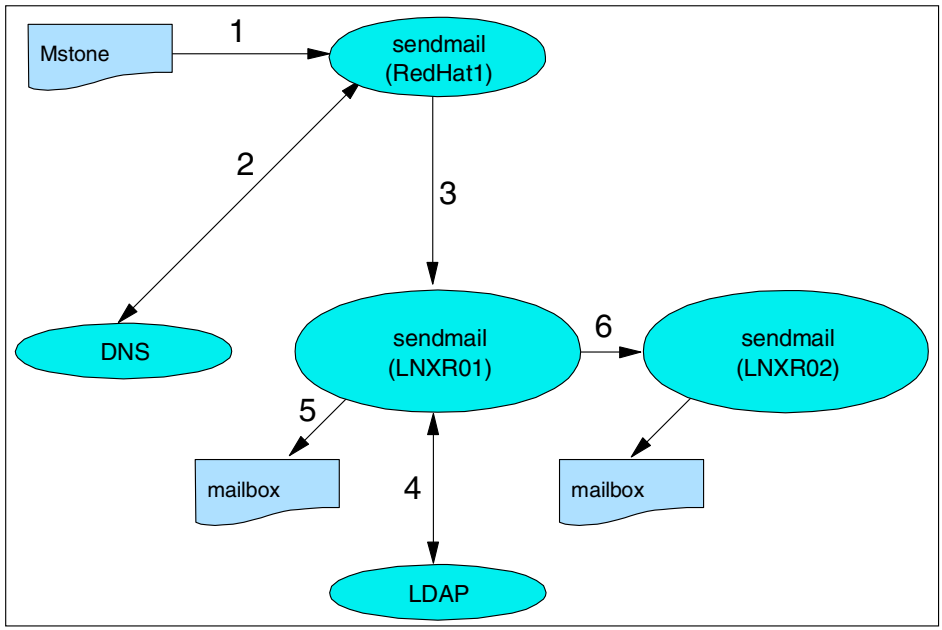


Figure B-2 Mstone workload flow

The operation proceeds as follows:

1. **Mstone creates a message to a randomly chosen example.com recipient.**
   The Mstone script randomly selects an intended recipient in the range itso-user0 through itso-user6999. The message is then sent to the local sendmail server (on host RedHat1 in the example) for delivery. Sendmail configuration for the Mstone clients is discussed in “Configuring the Mstone client” on page 164.
2. **The local sendmail server queries the example.com DNS server for the MX records required to deliver the message.**
   The Mstone client machines are configured to use the example.com DNS server (LNXR10) for host name resolution. When queried for MX records, DNS responds with the list of MX records that specify the mail exchange servers for the example.com domain (LNX01 through LNXR07).

   The first record in the list is selected by the DNS server in a round-robin fashion. This ensures messages are routed to a specific example.com sendmail server in a relatively equal manner.

3. **The local sendmail server forwards the message to the first mail exchange server identified by DNS.**
   The local sendmail server selects the specific mail exchanger server by proceeding through the MX record list in sequential order. The message is delivered to the first available sendmail server.

   Note: If the first mail exchange server in the list is unavailable (for example, that server is down), the local sendmail server attempts to deliver to the next mail exchange server in the list.

   Sendmail configuration for the example.com domain is discussed in "Configuring the example.com domain" on page 164.

4. **The selected example.com mail exchange server queries LDAP to determine where the recipient's mailbox resides.**
   Upon receipt of an incoming message, the specific example.com mail exchange server first queries LDAP with the intended recipient's e-mail address. LDAP responds with the host name on which that recipient's mailbox can be found. If the recipient's mailbox is defined locally, sendmail delivers the message to that user and processing is complete.

5. **If the recipient's mailbox is defined on another host, sendmail forwards the message to that host's mail exchange server.**
   In the event the recipient's mailbox is defined on another host, sendmail forwards the message to mail exchange server on that host.

LDAP routing in sendmail

As shown is Figure B-2 on page 162, the sendmail servers are configured to use LDAP when routing messages to real users. The choice of LDAP as a sendmail router has several advantages:

- Sendmail recipients can be defined without creating user IDs on the Linux hosts.
- The LDAP server adds an additional workload that can be studied and tuned in performance analysis.

Configuring sendmail to use LDAP routing is covered in "Configuring the example.com domain" on page 164. Additional information about LDAP routing for sendmail can be found at:

http://www.sendmail.org/m4/ldap_routing.html

Configuring the Mstone client

To configure the Mstone client:

1. **Install and start the sendmail server.**
   For Red Hat Linux installations, the default sendmail configuration is sufficient.
2. **Point the DNS host to the example.com DNS server.**
   Configure the resolver to point to the DNS server for the example.com domain. Add the following to the /etc/resolv.conf file:

```
search example.com
nameserver 172.22.10.22
```

Configuring the example.com domain

To configure the example.com domain for Mstone workloads, the sendmail, LDAP, and DNS server must be configured.

To enable sendmail for LDAP routing, we change the sendmail m4 configuration file (/etc/mail.sendmail.mc), as shown in Example B-1 on page 165.

Example: B-1 The sendmail configuration file: /etc/mail/sendmail.mc

```
divert(-1)
include(`/usr/share/sendmail-cf/m4/cf.m4')
VERSIONID(`linux setup for Red Hat Linux')dnl
OSTYPE(`linux')
define(`confDEF_USER_ID',``8:12'')dnl
define(`confDOMAIN_NAME', `lnxr01.example.com')dnl
undefine(`UUCP_RELAY')dnl
undefine(`BITNET_RELAY')dnl
define(`confAUTO_REBUILD')dnl
define(`confTO_CONNECT', `1m')dnl
define(`confTRY_NULL_MX_LIST',true)dnl
define(`confDONT_PROBE_INTERFACES',true)dnl
define(`PROCMAIL_MAILER_PATH',`/usr/bin/procmail')dnl
define(`ALIAS_FILE',
`ldap: -1 -v rfc822MailMember -k "(&(objectClass=nisMailAlias)(uid=%0))"')                1
define(`UUCP_MAILER_MAX', `2000000')dnl
define(`confUSERDB_SPEC', `/etc/mail/userdb.db')dnl
define(`confPRIVACY_FLAGS', `authwarnings,novrfy,noexpn,restrictqrun')dnl
define(`confLDAP_DEFAULT_SPEC',`-h 172.22.10.22 -b "o=My Organization Name,c=US"')        2
FEATURE(`no_default_msa',`dnl')dnl
FEATURE(`smrsh',`/usr/sbin/smrsh')dnl
FEATURE(`mailertable',`hash -o /etc/mail/mailertable.db')dnl
FEATURE(`virtusertable',`hash -o /etc/mail/virtusertable.db')dnl
FEATURE(redirect)dnl
FEATURE(always_add_domain)dnl
FEATURE(use_cw_file)dnl
FEATURE(use_ct_file)dnl
FEATURE(local_procmail,`',`procmail -t -Y -a $h -d $u')dnl
FEATURE(`access_db',`hash -o /etc/mail/access.db')dnl
FEATURE(`blacklist_recipients')dnl
EXPOSED_USER(`root')dnl
DAEMON_OPTIONS(`Port=smtp,Addr=0.0.0.0, Name=MTA')
FEATURE(`accept_unresolvable_domains')dnl
FEATURE(`relay_entire_domain')dnl
LDAPROUTE_DOMAIN(`example.com')                                                          3
FEATURE(`ldap_routing',
ldap -1 -v mailHost -k `"(&(objectClass=inetLocalMailRecipient) (mail=%0))"',
ldap -1 -v mailRoutingAddress -k `"(&(objectClass=inetorgperson) (mail=%0))"',
passthru)dnl                                                                             4
MAILER(smtp)dnl
MAILER(procmail)dnl
```

Configuration options are explained as follows:

1. The ALIAS_FILE directive instructs sendmail to convert a recipient name to a real user name using an LDAP query.

2. Identifies the LDAP server to use for mail routing (172.22.10.22) and provides the suffix attribute (`"o=My Organization Name,c=US"`) to append to LDAP queries.
3. Specifies that messages to users in the example.com domain are to be routed using LDAP.
4. Enables LDAP routing.

To generate a new sendmail configuration file, use the command:

```
m4 /etc/mail/sendmail.mc > /etc/sendmail.cf
```

Then restart the sendmail server.

The OpenLDAP server configuration file is shown in Example B-2.

Example: B-2 The OpenLDAP configuration file: /etc/openldap/slapd.conf

```
include/etc/openldap/schema/core.schema
include/etc/openldap/schema/cosine.schema
include/etc/openldap/schema/inetorgperson.schema
include/etc/openldap/schema/nis.schema
include/etc/openldap/schema/redhat/autofs.schema
include/etc/openldap/schema/redhat/kerberosobject.schema
include/etc/openldap/schema/misc.schema

#######################################################################
# ldbm database definitions
#######################################################################

loglevel0
databaseldbm
cachesize7500
dbcachesize600000
suffix "o=My Organization Name,c=US"
rootdn "cn=Manager,o=My Organization Name,c=US"
rootpw secret

directory/var/lib/ldap
# Indices to maintain
indexobjectClass,uid,uidNumber,gidNumber,memberUideq
indexcn,mail,surname,givennameeq,subinitial
```

Example B-3 on page 167 shows a partial listing of the /etc/named.conf DNS configuration file. The DNS server is defined to be authoritative for the example zone as indicated by the `type master` option.

Example: B-3 DNS configuration for zone example.com: /etc/named.conf

```
.
.
zone "example.com" {
        type master;
        notify yes;
        file "db.com.example";
};
.
.
```

The /var/named/db.example.conf file shown in Example B-4 defines the example.com zone.

Example: B-4 The forward DNS zone configuration:/var/named/db.example.com

```
$TTL 86400
@               IN      SOA     lnxr10.example.com. root.example.com. (
                2003011816 86400 7200 604800 86400 )
                IN      NS      lnxr10
                IN      MX      10 lnxr07
                IN      MX      10 lnxr06
                IN      MX      10 lnxr05
                IN      MX      10 lnxr04
                IN      MX      10 lnxr03
                IN      MX      10 lnxr02
                IN      MX      10 lnxr01

lnxr01 IN A  172.22.10.13
lnxr02 IN A  172.22.10.14
lnxr03 IN A  172.22.10.15
lnxr04 IN A  172.22.10.16
lnxr05 IN A  172.22.10.17
lnxr06 IN A  172.22.10.18
lnxr07 IN A  172.22.10.19
lnxr08 IN A  172.22.10.20
lnxr09 IN A  172.22.10.21
lnxr10 IN A  172.22.10.22
```

Example B-5 illustrates round-robin balancing when querying mail exchange records.

Example: B-5 Round-robin DNS mail exchange

```
$ host -t mx example.com 172.22.10.22
Using domain server:
Name: 172.22.10.22
Address: 172.22.10.22#53
Aliases:

example.com mail is handled by 10 lnxr04.example.com.
example.com mail is handled by 10 lnxr05.example.com.
example.com mail is handled by 10 lnxr06.example.com.
example.com mail is handled by 10 lnxr07.example.com.
example.com mail is handled by 10 lnxr01.example.com.
example.com mail is handled by 10 lnxr02.example.com.
example.com mail is handled by 10 lnxr03.example.com.
$ host -t mx example.com 172.22.10.22
Using domain server:
Name: 172.22.10.22
Address: 172.22.10.22#53
Aliases:

example.com mail is handled by 10 lnxr01.example.com.
example.com mail is handled by 10 lnxr02.example.com.
example.com mail is handled by 10 lnxr03.example.com.
example.com mail is handled by 10 lnxr04.example.com.
example.com mail is handled by 10 lnxr05.example.com.
example.com mail is handled by 10 lnxr06.example.com.
example.com mail is handled by 10 lnxr07.example.com.
```

Populating the LDAP database

To populate the LDAP database, we add sendmail users using an LDAP Data Interchange Format (LDIF) import file. A portion of the LDIF definition is shown in Example B-6 on page 169.

Example: B-6 The LDIF definition of LDAP sendmail users: users.ldif

```
dn: uid=itso-user0, o=My Organization Name,c=US                    1
userpassword: pwd
givenname: itso-user0
sn: itso-user0
cn: itso-user0
uid: itso-user0
uidNumber: 1000
gidNumber: 1000
mail: itso-user0@example.com
mailhost: lnxr01.example.com
homeDirectory: /home/user/itso-user0
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: inetLocalMailRecipient
objectclass: posixAccount

dn: uid=allusers, o=My Organization Name,c=US                      2
userpassword: pwd
givenname: allusers
sn: allusers
cn: allusers
uid: allusers
mail: allusers@example.com
rfc822MailMember: itso-user0@example.com
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: nisMailAlias
```

For each user, we add distinguished name (DN) entries:

1. A DN entry for the user in the form `itso-user`*n*

   This entry defines a real sendmail user. As part of the definition, the user is assigned to a specific mailhost (in this case, lnxr01.example.com).

2. A DN entry for the user `allusers`

   Each user is added to the allusers alias. Mail addresses to allusers@example.com is delivered to every sendmail user (itso-user0 through itso-user6999).

To add the users to the LDAP database, use the command:

```
ldifadd -D "cn=Manager,o=My Organization Name,c=US" -W -x -f users.ldif
```

C

Performance Toolkit for VM

This appendix describes some of the Linux monitoring features available with the IBM Performance Toolkit for VM.

Linux monitoring with the Performance Toolkit for VM

Retrieval and display of Linux internal performance data is based on the Linux DDS server interface, originally written for use with the Resource Management Facility (RMF™) PM. To monitor Linux internal performance, the following components are required:

- The toolkit must be installed, configured, and active.
  Performance Toolkit for VM, an optional feature of z/VM V4.4 (5739-A03).
- DDS interface must be installed and active; for further information, access:

  http://www.ibm.com/servers/eserver/zseries/zos/rmf/rmfhtmls/pmweb/pmlin.htm
- Performance data must be collected, stored, and managed on the Linux system.

After these are active, specific information that can be provided includes:

- Linux system details:
  - Processes created per second
  - Context switches per second
  - Apache:
    - Requests per second
    - Bytes per request
    - Busy threads
    - Idle threads
    - 404 error rate
- For each Linux system:
  - Linux CPU utilization, both user and kernel
  - CPU utilization by processor
  - CPU utilization by process
- For each Linux system:
  - Linux memory utilization
  - Total memory size
  - Memory in use
  - Memory use by process

- For each Linux system:
  - Linux network activity
  - Packets received and sent per second
  - Bytes received and sent per second
  - Receive and send error rates
- For each Linux system:
  - Linux file system usage
  - I/O request rates, response times
  - File system size, including megabytes free, percent used, and percent free

Additional, specific details about the installation and use of the toolkit are included in *z/VM: Performance Toolkit*, SC24-6062.

Abbreviations and acronyms

CCW	channel command word
CHPID	channel-path identifier
CMS	Conversational Monitor System
CP	Control Program
CP	Central Processor
CTC	Channel-to-channel
DASD	direct access storage device
DNS	Domain Name System
DPA	dynamic paging area
ECKD	extended count key data
ESCON	Enterprise Systems Connection
ESS	Enterprise Storage Server
FBA	fixed block architecture
FCP	Fibre Channel Protocol
FICON	Fibre Connection
FTP	File Transfer Protocol
IBM	International Business Machines Corporation
ICF	Internal Coupling Facility
IFL	Integrated Facility for Linux
IML	Initial Machine Load
IQDIO	Internal Queued Direct Input Output
ITSO	International Technical Support Organization
IUCV	Inter-User Communications Vehicle
LAN	local area network
LDAP	Lightweight Directory Access Protocol
LPAR	logical partition
LVM	Logical Volume Manager
MDC	minidisk cache
NSS	named saved system
OSA	Open Systems Adapter
PAV	Parallel Access Volume
QDIO	Queued Direct Input Output
RAID	redundant array of independent disks
RAMAC	RAID Architecture with Multi-Level Adaptive Cache
RDR	Reader
RVA	RAMAC Virtual Array
SCSI	small computer system interface
SNMP	simple network management protocol
SSCH	Start Subchannel
WPBS	WebSphere Performance Benchmark Sample

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 180.

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

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

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

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

- *Linux on IBM @server zSeries and S/390: ISP/ASP Solutions*, SG24-6299

  http://www.ibm.com/redbooks/abstracts/sg246299.html

- *Linux on IBM @server zSeries and S/390: Large Scale Linux Deployment*, SG24-6824

  http://www.ibm.com/redbooks/abstracts/sg246824.html

- *Linux on IBM @server zSeries and S/390: System Management*, SG24-6820

  http://www.ibm.com/redbooks/abstracts/sg246820.html

- *Tuning IBM @server xSeries Servers for Performance*, SG24-5287

  http://www.ibm.com/redbooks/abstracts/sg245287.html

- *WebSphere Application Server V4 for Linux, Implementation and Deployment Guide*, REDP0405

  http://www.ibm.com/redbooks/abstracts/redp0405.html

- *Enterprise Systems Connection (ESCON) Implementation Guide*, SG24-4662

  http://www.ibm.com/redbooks/abstracts/sg244662.html

- *FICON Native Implementation and Reference Guide*, SG24-6266

  http://www.ibm.com/redbooks/abstracts/sg246266.html

- *IBM @server zSeries Connectivity Handbook*, SG24-5444

  http://www.ibm.com/redbooks/abstracts/sg245444.html

- *Getting Started with zSeries Fibre Channel Protocol*, REDP0205

  http://www.ibm.com/redbooks/abstracts/redp0205.html
- *Linux on IBM @server zSeries and S/390: TCP/IP Broadcast on z/VM Guest LAN*, REDP3596

  http://www.ibm.com/redbooks/abstracts/redp3596.html
- *Implementing Fibre Channel Attachment on the ESS*, SG24-6113

  http://www.ibm.com/redbooks/abstracts/sg246113.html

Other resources

These publications are also relevant as further information sources:

- *z/VM V4R3.0 Performance*, SC24-5999
- *z/VM V4R3.0 CP Planning and Administration*, SC24-6043
- *z/VM V4R3.0 Virtual Machine Operation*, SC24-6036
- *z/VM V4R3.0 System Operation*, SC24-6000
- *z/VM V4R3.0 Running Guest Operating Systems*, SC24-5997
- *z/VM V4R3.0 CP Command and Utility Reference*, SC24-6008
- *z/VM: Performance Toolkit*, SC24-6062
- *Linux for zSeries and S/390 Device Drivers and Installation Commands*, LNUX-1303

Referenced Web sites

These Web sites are also relevant as further information sources:

- z/VM Performance Resources

  http://www.vm.ibm.com/perf/
- IBM developerWorks Linux for zSeries and S/390 home page

  http://www-124.ibm.com/developerworks/oss/linux390/index.shtml
- Velocity Software performance tips

  http://linuxvm.com
- Linux for S/390 home page

  http://linuxvm.org
- Samba dbench Benchmarking Tool

  http://samba.org/ftp/tridge/dbench/

- IBM WebSphere Performance Benchmark Sample (Trade 2)

  http://www.ibm.com/software/webservers/appserv/wpbs_download.html
- *Page replacement in Linux 2.4 memory management* by Rik van Riel

  http://www.surriel.com/lectures/linux24-vm.html
- Chris Gould's Linux Kernel Architecture and Other OS Links

  http://cs.uml.edu/~cgould
- *Understanding the Linux Virtual Memory Manager* by Mel Gorman

  http://www.csn.ul.ie/~mel/projects/vm/guide/html/understand/
- The Linux memory management home page

  http://linux-mm.org/
- Performance considerations for Linux guests

  http://www.vm.ibm.com/perf/tips/linuxper.html
- Configuring processor storage

  http://www.vm.ibm.com/perf/tips/storconf.html
- How to use VM shared kernel support

  http://www.vm.ibm.com/linux/linuxnss.html
- *z/VM, VSE, and Linux Technical Conference foils: z/VM Resource Management*, by Christine Casey

  http://www.vm.ibm.com:2003/pdfs/V612up.pdf
- Bonnie benchmark home page

  http://www.textuality.com/bonnie/
- Data Test program (dt) home page

  http://www.bit-net.com/~rmiller/dt.html
- Iperf TCP and UDP bandwidth measurement tool

  http://dast.nlanr.net/Projects/Iperf/
- The Mstone performance tool

  http://www.mozilla.org/projects/mstone
- Mailstone utility documentation

  http://docs.sun.com/source/816-6036-10/index.html
- LDAP routing for sendmail

  http://www.sendmail.org/m4/ldap_routing.html

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Linux on IBM @server zSeries and S/390:

Performance Measurement and Tuning

Redbooks

Understanding Linux performance on zSeries

z/VM performance concepts

Tuning z/VM Linux guests

This IBM Redbook examines performance measurement and tuning for running Linux as a z/VM guest on IBM @server zSeries and S/390 machines. This publication is intended for system administrators and I/T architects responsible for deploying Linux servers running under z/VM. We examine performance concepts and identify tuning parameters that influence system performance. Using examples, we investigate performance and tuning topics for the memory, processor, DASD, and networking subsystems. Performance is analyzed at both the z/VM and Linux level. We provide tuning recommendations and guidance intended to maximize investment in Linux for zSeries.

The system used in this writing the redbook is an IBM @server zSeries 900 running z/VM Version 4.3 in an LPAR. The Linux distributions used in this redbook include Red Hat Version 7.2 for zSeries (based on a Linux 2.4.9 kernel) and SuSE SLES7 (based on a Linux 2.4.7 kernel).

The intent of this redbook is to provide guidance on measuring and optimizing performance using an existing zSeries configuration. The examples are intended to demonstrate how to make effective use of your zSeries investment. The workloads used are chosen to exercise a specific subsystem; any measurements provided should not be construed as a benchmark.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6926-00 ISBN 0738429473