Linux on IBM System z Performance Measurement and Tuning

Understanding Linux performance on System z

z/VM performance concepts

Tuning z/VM Linux guests

Lydia Parziale
Edi Lopes Alves
Linda Caroll
Mario Held
Karen Reed

Redbooks

ibm.com/redbooks

International Technical Support Organization

Linux on IBM System z: Performance Measurement and Tuning

December 2011

SG24-6926-02

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

Third Edition (December 2011)

This edition applies to Version 6, Release 1 of z/VM RSU 1003 and Linux SLES11 SP1.

Contents

Notices

Trademarks

Preface

This IBM® Redbooks® publication discusses performance measurement and tuning for Linux for System z®. It is intended to help system administrators responsible for deploying Linux for System z understand the factors that influence system performance when running Linux as a z/VM® guest.

This book starts by reviewing some of the basics involved in a well-running Linux for System z system. An overview of some of the monitoring tools that are available and some that were used throughout this book is also provided.

Additionally, performance measurement and tuning at both the z/VM and the Linux level is considered. Some tuning recommendations are offered in this book as well. Measurements are provided to help illustrate what effect tuning controls have on overall system performance.

The system used in the writing of this book is IBM System z10® running z/VM Version 6.1 RSU 1003 in an LPAR. The Linux distribution used is SUSE Linux Enterprise Server 11 SP1. The examples in this book use the Linux kernel as shipped by the distributor.

The z10 is configured for:

Main storage	6 GB
Expanded storage	2 GB
Minidisk cache (MDC)	250 MB
Total LPARs	45
Processors	Four shared central processors (CPs) defined as an uncapped logical partition (LPAR)

The direct access storage devices (DASD) used in producing this IBM Redbooks publication are 2105 Enterprise Storage Server® (Shark) storage units.

The intent of this book 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 benchmarks.

The team who wrote this book

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

Lydia Parziale is a Project Leader for the ITSO team in Poughkeepsie, New York, with domestic and international experience in technology management including software development, project leadership, and strategic planning. Her areas of expertise include e-business development and database management technologies. Lydia is a Certified IT Specialist with an MBA in Technology Management and has been employed by IBM for 24 years in various technology areas.

Edi Lopes Alves is an IT Systems Management Specialist with IBM Global Services, Brasil. She has more than 20 years of experience as VM systems programmer and IBM DB2®

Content Manager solutions in Finance area. Edi is a certified z/Series Specialist with a Masters degree in E-business from ESPM in Sao Paulo. She currently supports IBM z/VM and LINUX in IBM Global Accounts (IGA) and her area of expertise is the general performance of Linux on System z,

Linda Caroll is an IT Specialist with IBM ITD, Delivery Technology and Engineering and is based in Atlanta, Georgia. She has over thirty years of experience in System z specializing in capacity management. Linda has presented papers at SHARE and CMG on the subject of capacity planning and methodology. She has developed a forecasting methodology that uses seasonality with linear trending. She has been with IBM for thirteen years and prior to IBM worked in the insurance, health care, retail and credit industries.

Mario Held is a Software Performance Analyst for the Linux on System z - System & Performance Evaluation Development in the IBM development lab in Boeblingen, Germany, since 2000. His area of expertise is the general performance of Linux on System z, and he specializes in gcc and glibc performance. He presents worldwide on all areas of performance. Between 1997 and 2000 he was a System Developer. For six years before joining the IBM development lab he was an application programer on the mainframe with a health insurance company in Germany.

Karen Reed is an IBM Senior Systems Engineer supporting IBM Tivoli® software in San Francisco, California. She has over twenty years of experience in systems performance tuning and automation software for both System z and distributed systems. Karen is experienced in architecture design and implementation planning for complex and closely coupled systems. Her performance and capacity planning expertise covers processor resource allocation, I/O subsystems, operating systems and applications.

Figure 1 The team who wrote this book (l-r): Edi Lopes Alves, Karen Reed, Mario Held, and Linda Caroll

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

Roy P. Costa
International Technical Support Organization, Poughkeepsie Center

Bill Bitner
z/VM Software Performance Analyst, z/VM Development, Endicott

Christof Schmitt
SONAS System Development, IBM USA

Special thanks to our technical review team:

Ursula Braun, Jan Glauber, Stefan Haberland, Martin Kammerer, Mustafa Mesanovic, Gerald Schaefer, Martin Schwidefsky, and Steffen Maier

Additional thanks to the authors of the release 2 of this book. Authors of the second edition, *Linux on IBM System z: Performance Measurement and Tuning*, published in February 2008, were Massimiliano Belardi, Mario Held, Li Ya Ma, Lester Peckover, Karen Reed

Additional thanks to the authors of the first edition of this book. Authors of the first edition, *Linux on IBM System z: Performance Measurement and Tuning*, published in May 2003, were Gregory Geiselhart, Laurent Dupin, Deon George, Rob van der Heij, John Langer, Graham Norris, Don Robbins, Barton Robinson, Gregory Sansoni, and Steffen Thoss.

Comments welcome

Your comments are important to us!

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

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

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

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

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-6926-02
for Linux on IBM System z: Performance Measurement and Tuning
as created or updated on December 27, 2011.

December 2011, Third Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- Added information about capacity planning
- Added information about new supported features in Linux kernels up to 2.6.32
  - Large page support
  - CPU topology support
  - cpuplugd service
- Added information about new DS8000® features
  - Storage Pool Striping (SPS)
- Added information about IBM Tivoli OMEGAMON® agentless monitoring for Linux
- Added information about the CPU SHARE command
- Performance implications using mixed engine

Changed information

- Updated information specific to z/VM to the current release of 6.1
  - Emergency scans updated
  - Page reorder issue
- Updated information regarding new Linux on System z features
  - Changes in supported network drivers and parameters
  - Changes with disk I/O statistics
- Removed information regarding Infrastructure cost
- Removed all hints related to the 2 GB line constraint of previous z/VM versions
- Removed references to the *on demand timer patch*
- Updated information about monitoring software products
- Updated information about LPARs, CPU Timer Functionality

February 2008, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- Moved all of the Linux monitoring tools to their own chapter
  - Added Omegamon XE for z/VM and Linux
  - Added performance analysis tools such as SAR, vmstat, and iostat
- Moved tuning to its own chapter
  - Added bottleneck analysis
  - Added sizing considerations
  - Added benchmark concepts and practices
- Added information about emergency scanning
- Added information about new disk and network features

Changed information

- Updated information specific to z/VM to the current releases of v5.2 and v5.3
- Updated information specific to Linux kernel 2.6

1

Virtualization and server consolidation

In this chapter, we define virtualization and discuss benefits gained by running Linux as a z/VM guest. We examine sharing real hardware resources using the virtualization technology provided by z/VM.

1.1 Server consolidation and virtualization

The concept of virtualization is quickly becoming commonplace in data centers around the world. Generally, virtualization describes a shared pool resource environment where large numbers of virtual servers are consolidated onto a small number of relatively large physical servers. This is opposite of the more common environment where each workload is run on discrete servers with non-shared resources. The System z mainframe is ideally suited for this large hosting server role.

Selecting workloads suitable for virtualization should be done during early in the planning stage. Any application that relies heavily on a single resource (for example, a CPU with relatively low I/O and memory requirements) is not a good candidate for virtualization. Commonly encountered workloads that use CPU, I/O, and memory in a more moderate, balanced fashion over time are good candidates, as they can coexist more readily in a shared resource infrastructure.

Certain workloads involve mainly number crunching activity, for example, financial forecasting applications.

We need to be careful of misleading comparisons that consider only one of the major resources. If we compare the raw cycle speed of a System z CPU with an average Pentium processor, or a single processor, on a multi-core processor chip, the comparison might suggest that System z servers are not an advantage. If that workload uses 100% of a modern PC for 24 hours a day, the same workload could easily use most of a System z CPU for the entire day. Virtualizing this workload means effectively dedicating part of the CPU resource to a single workload, which is not conducive in a virtualized, shared pool environment.

A System z machine has a higher maximum number of processors than any other platform. A computing-intensive application would run well, though it is not the optimum candidate for System z hardware because it does not take advantage of the virtualization and hardware sharing capabilities. Such extremes are not typical, and there are many situations where using System z does make a lot of sense. In this book, we show a case study that uses an IBM WebSphere workload generated by the Trade 6 Benchmark to represent a typical balanced workload that is well suited to virtualization.

Consider a case where the workload uses the machine for 12 hours instead of 24. Imagine that there is another similar workload using the same amount of resources, but during the other 12 hours of the day. In this situation, we can put both Linux systems on the same System z and let them use all the CPU cycles that they need. These two systems would easily coexist on the same hardware, running as virtual server guests using approximately half of the purchased CPU power than if they were running on freestanding, discrete hardware. We show examples of such scenarios later in this book.

Lowering total cost of ownership (TCO) is another advantage of using a virtualized environment on the mainframe. Taking all factors into account, including hardware cost, software licensing, on-going support costs, space usage, and machine room running and cooling costs, the System z option delivers the overall lowest TCO for installations with a currently large investment comprising many distributed servers. The reliability, availability, and serviceability that are proven strengths of the mainframe are additional positive factors.

When multiple Linux systems run on a System z, each Linux system acts as if it has dedicated access to a defined portion of the System z machine, using a technique known as timesharing. Each Linux system runs in its own virtual machine whose characteristics (for example, memory size and number of CPUs) define the hardware that Linux sees. The allocation and tuning controls in z/VM specify how real hardware resources are allocated to the virtual machine.

1.1.1 Virtualization of the CPU

Processor virtualization is accomplished by timesharing, meaning that each Linux guest, in turn, gains access to a processor for a period of time. As each time period is completed, the processor is available for other Linux systems. This cycle continues over time, allowing each system access as needed.

The cost of running both these workloads on discrete servers is twice the cost for running one workload. However, by sharing the resources effectively with System z, we can run both workloads for the price of one. Real-world business applications in practice often run less than 50% of the time, which allows you to run more of these workloads on the same System z machine. The workload from these business applications is most likely not a single, sustained burst per day, but rather multiple short bursts over the entire day. It is possible to spread the total processor requirements over the full day given enough servers and short workload intervals.

Because the System z is specifically designed for timesharing, z/VM can switch between many system tasks in a very cost-effective manner. This role has been perfected for decades so that the z/VM we see today contains the refinement and high levels of reliability demanded by modern information technology (IT).

1.1.2 Virtualization of memory

In addition to the CPU requirements, workloads have memory requirements. The memory needed to contain and run the Linux system is the working set size. Memory virtualization on z/VM is done using paging, a technique that places the memory in a central shared pool in the machine where Linux systems take turns using the main storage, or main memory. Paging volumes reside on disks. Pages of inactive Linux systems are held in expanded storage and can be quickly moved into z/VM main storage when needed.

The challenge for z/VM is to bring in the working set of a Linux system as quickly as possible whenever there is a work request. z/VM answers this challenge easily due to its central and expanded storage architecture and advanced I/O subsystem.

1.1.3 Virtualization of the network

Network resources (for example, IP addresses, network adapters, LANs, and bandwidth management) must be allocated whenever applications and servers need to connect to other servers. Network resources can be virtualized, or pooled and shared, in z/VM. This facilitates faster, more efficient, cost-effective, secure and flexible communication across the entire IT infrastructure while eliminating outages due to physical or software network device failures.

Virtual LAN (VLAN) provides a network infrastructure inside z/VM by furnishing the isolation required to ensure that data will not be interspersed while flowing across the shared network. Guests in the same VLAN can communicate directly, with cross-memory speed, without outside box routing or intermediate hardware thus eliminating latency.

1.1.4 Levels of virtualization

The virtual machine provided by z/VM to run the Linux system is not the only virtualization that is taking place. The Linux system itself runs multiple processes, and the operating system allocates resources to each of these processes. Certain processes running on Linux run multiple tasks and allocate their resources to those tasks. If the z/VM system runs in a logical partition (LPAR), which again uses the timesharing principle, z/VM also uses only part of the real hardware.

These multiple layers of virtualization can make it hard for an operating system to find the best way to use the allocated resources. In many cases, tuning controls are available to solve such problems, and they allow the entire system to run efficiently. When the operating system was not originally designed to run in a shared environment, certain controls turned out to have surprisingly negative side effects.

1.1.5 Benefits of virtualization

Management's primary concern is to increase revenue while reducing costs. But as IT systems grow larger and more complex, the cost of managing systems rises faster than the cost of purchasing system hardware. Virtualization increases management efficiency by lowering costs (compared to existing infrastructure), reducing complexity, and improving flexibility to quickly respond to business needs.

Virtualization offers the benefits listed below by consolidating hundreds of existing under-utilized and unstable servers from their production and development environment to a single, or a small number, of System z boxes:

- Higher resource utilization

  Running as guests on z/VM, several Linux servers can share the physical resources of an underlying box, resulting in higher levels of sustained resource utilization, comfortable even when approaching 100% for processor. This is especially relevant for variable workloads whose average needs are much less than entire dedicated resources and who do not peak at the same time. Two workloads that only run during different halves of the day are a good example.

- More flexibility

  Linux servers can dynamically reconfigure resources without having to stop all of the running applications.

- Improved security and guest isolation

  Each z/VM virtual machine can be completely isolated from the control program (CP) and other virtual machines, so if one virtual machine fails the others are not affected. Data is prevented from leaking across virtual machines, and applications can communicate only over configured network connections.

- Higher availability and scalability

  Linux servers running on z/VM can improve their availability due to the reliability of underlying mainframes. These systems have refined hardware and software architectures that have evolved over four decades. Physical resources can be removed, upgraded, or changed without affecting their users. z/VM can easily scale Linux guests to accommodate changing workload demands.

- Lower management costs

  Consolidating a number of physical servers onto a single or vastly smaller number of larger processors reduces infrastructure complexity and concentrates and automates common management tasks, thus improving productivity.

- Improved provisioning

  Because virtual resources are abstracted from hardware and operating system issues, they are capable of recovering more rapidly after a crash. Running in this virtualized mode greatly facilitates high-speed cloning, allowing extra guests to be easily created and made available in seconds.

1.2 Sharing resources

A shared resource works in the following ways:

- Multiple virtual machines take turns using a resource.

  This generally occurs when the processor is shared. z/VM dispatches virtual machines one after the other on the real processor. The required resources are provided so effectively that each virtual machine believes that 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. Because some work is needed to manage and allocate the resources, this type of sharing is not free. There is an overhead in z/VM for switching back and forth between virtual machines. The amount of this overhead depends on the number of virtual machines competing for the processor, but it is a relatively small part of the total available resources on a properly tuned system.

- z/VM allows virtual machines to share memory pages.

  Here 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 addition to saving memory resources, this has important implications when servicing the system, as only a single change is necessary to update all of the sharing users. In z/VM, this is done through named saved systems (NSS). It is possible to load the entire 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 residing in their private working set. There is no additional cost, in terms of memory resources, for z/VM when more virtual machines start to share the NSS. Note that not all memory sharing is managed by using NSS. The virtual machines running in the z/VM layer also compete for main memory to hold their private working set. The NSS typically holds only a relatively small part of the total working set of the Linux virtual machine.

1.2.1 Overcommitting resources

It is possible to overcommit your hardware resources. For example, when you run 16 virtual machines with a virtual machine size of 512 MB in a 4 GB z/VM system, you overcommit memory approximately by a factor of two. This works as long as the virtual machines do not require the allocated virtual storage at exactly the same time, which is often the case.

However, overcommitting resources can be beneficial. A restaurant, for example, could be seat 100 customers and allow all of them to use the restrooms. The service can be provided with only a small number of restrooms. However, a theater needs more restrooms per 100 people because of the expected usage pattern, which tends to be governed by the timing factors such as movie start and end. This shows how the workload and the time that it runs affect the ability to share a hardware resource. This example also shows that partitioning your resources (for example, separate restrooms for male and female guests) reduces your capacity if the ratio between the workloads is not constant. Other requirements, such as service levels, might require you to do so anyway. Fortunately, the z/VM system has the flexibility to cope well with similar practical situations within a computing system.

When the contention on the shared resources increases, the chances of queuing can increase, depending on the total resources available. This is one consequence of sharing that cannot be avoided. When a queue forms, the requesters are delayed in their access to the shared resources to a variable extent. Whether such a delay is acceptable depends on service levels in place on the system and other choices that you make.

However, overcommitting is generally necessary to optimize the sharing of a resource. Big problems can arise when all resources in the system are overcommitted and required at the same time.

1.2.2 Estimating the required capacity

Some benchmarks measure the maximum throughput of an application and determine the maximum possible number of transactions per second, the number of floating point operations per second, and the number of megabytes transferred per second for an installed system. The maximum throughput of the system, when related to anticipated business volumes, will help you plan capacity requirements of the system.

Applications must be efficient and run well on the installed hardware base because multiple virtual machines compete on z/VM for resources. Typical metrics for these measurements are based upon a unit rate relation between workload and time, such as megabytes transferred per CPU second, or the number of CPU seconds required per transaction.

2

Tuning basics on System z

Every hardware and software platform has unique features and characteristics that must be considered when you tune the environment. System z processors have been enhanced to provide robust features and high reliability. This chapter discusses general tuning methodology concepts and explores their practical application on a System z with Linux guests. Topics covered in this chapter include processor sharing, memory, Linux guest sizing, and benchmark methodology.

2.1 The art of tuning a system

Performance analysis and tuning is a multi-step process. Regardless of which tools you choose, the best methodology for analyzing the performance of a system is to start from the outside and work your way down to the small tuning details. Start by gathering data about the overall health of systems hardware and processes. How busy is the processor during the peak periods of each day? What happens to I/O response times during those peaks? Do they remain fairly consistent, or do they elongate? Does the system get memory constrained every day, causing page waits? Can current system resources provide user response times that meet service level agreements? Following a good performance analysis process can help you answer these questions.

It is important to know what tuning tools are available and what type of information they provide (see Chapter 3, “z/VM and Linux monitoring tools” on page 13). Equally important is knowing when to use those tools and what to look for. Waiting until the telephone rings with user complaints is too late to start running tools and collecting data. How will you know what is normal for your environment and what is problematic unless you check the system activity and resource utilization regularly? Conducting regular health checks on a system also provides utilization and performance information that you can use for capacity planning.

A z/VM system offers many controls (that is, tuning knobs) to influence the way that resources are allocated to virtual machines. Few z/VM controls increase the amount of resources available. In most cases, the best that can be done is to 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 is not a one-size-fits-all approach, as a system tuned for one type of workload performs poorly with another type of workload. This means that you must understand the workload that you want to run and be prepared to review your tuning efforts when the workload changes.

2.1.1 What tuning does not do

Understand that you cannot run more work than can fit in the machine. For example, if your System z machine has two processors and you want to run a workload of three Linux virtual machines running WebSphere®, with each running a CPU for 100% all day, then your workload will not fit. Tuning the z/VM will not improve the fit, but there might be performance issues in the applications that can change the workload to use less than 100% all day.

Conversely, no tuning is necessary if a System z machine with four processors runs a workload of three Linux virtual machines, each using a processor at 100% all day. Here the z/VM has sufficient resources to give each Linux virtual machine what it requires. Even so, you can change the configuration to use all four processors and thus increase running speed.

2.1.2 Where tuning can help

The system might not perform as expected even when various workloads add up to less than the total amount of resources available, possibly because the system is short on one specific resource. Tuning can make a difference in such situations. Before you begin tuning, determine what resource is the limiting factor in your configuration. Tuning changes fall into the following categories:

- Use less constrained resources.

  A benefit of running Linux systems under z/VM is the ability to share and overcommit hardware resources. The amount of memory that Linux thinks it owns is called virtual memory. The sum of the amounts of virtual memory allocated to each Linux guest can be many times the amount of real memory available on the processor. z/VM efficiently manages memory for Linux guests. When a system is memory constrained, one option is to reduce overall z/VM memory usage by reducing the virtual machine size of Linux guests.

- Get a larger share of a constrained resource.

  You can easily increase the virtual memory size for Linux guests by changing the guest definition under z/VM. Keep in mind that needlessly increasing virtual memory allocations can cause excessive paging, or thrashing, for the entire system. If a system is truly memory constrained and Linux virtual memory sizes have been assigned judiciously, consider reserving memory pages for one Linux virtual machine at the expense of all others. Do this with the z/VM command **`cp set reserved`**.

- Increase total available resources.

  The most obvious way to solve memory issues is to add more hardware, a viable option because memory costs have declined. You can also add resources by stopping unneeded utility services in the Linux systems. Remember that tuning does not increase the total amount of system resources, but rather allows those resources to be used more effectively for critical workloads.

2.1.3 Exchanging resources

Tuning is the process of exchanging one resource for another where configuration changes direct an application to use less of one resource and more of another. If you consider IT budget and staff hours as a resource, purchasing additional processors is also an exchange of one resource for another.

Consider the WebSphere benchmark sample workload (discussed in Appendix A, “WebSphere performance benchmark sample workload” on page 219). This is an end-to-end benchmark configuration with a real-world workload driving WebSphere’s implementation of J2EE web services. It runs in a multi-tiered architecture using three Linux systems:

- WebSphere runs in the first.
- The IBM HTTP server runs in a second.
- DB2 runs in the third Linux.

Running multiple virtual machines can increase costs because duplicating the Linux operating system and infrastructure requires additional communication between virtual machines, and certain features that could be shared are being duplicated. However, you can tune the resources given to each virtual machine. For example, you can set the web server small enough to run quickly and come into memory easily. You can set the database virtual machine larger so to cache large amounts of data and not require as much I/O to disk.

z/VM can dispatch these virtual machines on real processors independently. This allows the WebSphere server to use more processor cycles because it does not have to wait for the database storage to come in. Set everything as large as possible and use it to the maximum in an unconstrained environment.

2.1.4 Workload profile

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

The most cost-efficient approach to running these workloads is to adjust the capacity of the server during the day. This is exactly what z/VM carries out. Portions of the virtual machine are brought in to run in main memory while inactive virtual machines are moved to paging to create space.

2.2 Determining the problem

Determining a problem requires the skills of a systems performance detective. A systems performance analyst identifies IT problems using a detection process similar to that of solving a crime. In IT systems performance, the crime is a performance bottleneck or sudden degrading response time. The performance analyst asks questions, searches for clues, researches sources and documents, reaches a hypothesis, tests the hypothesis by tuning or other means, and eventually solves the mystery, which results in improved system performance.

2.2.1 Where to start

Begin with asking the IT staff questions:

- When did the problem first occur?
- What changes were made to software, hardware, or applications?
- Is the performance degradation consistent or intermittent?
- Where is the problem occurring?
- What areas does the problem affect?

You can also use the performance tools discussed in Chapter 3, “z/VM and Linux monitoring tools” on page 13.

2.2.2 Steps to take

Bottleneck analysis and problem determination are facilitated by sophisticated tools such as IBM Tivoli OMEGAMON on z/VM and Linux. OMEGAMON detects performance problems and alerts you before degraded response time becomes evident. We use OMEGAMONI in the following bottleneck analysis scenario.

Bottleneck example:

1. OMEGAMON detects a potential performance or availability problem and sends a warning alert for z/VM.
2. The OMEGAMON console contains a tree-structured list of all monitored systems and applications and displays an initial warning icon (a yellow triangle with an exclamation mark) next to the affected z/VM system and the area of concern. Real Storage is the problem area shown in Figure 2-1.

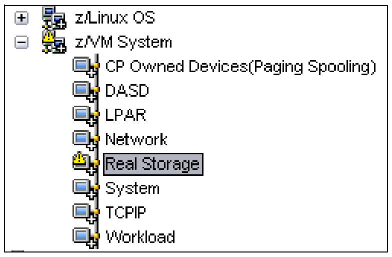
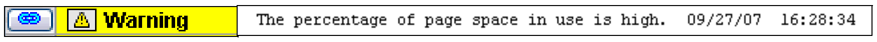


Figure 2-1   Warning icon

3. Click the warning icon to see details. In our example, the percentage of paging space in use is high (Figure 2-2).



```
The percentage of page space in use is high.  09/27/07  16:28:34
```

Figure 2-2   Additional warning information

4. The warning message displays the problem. Click the link icon to the left of the warning to see more details and possible solutions.
5. If the problem persists, OMEGAMON sends a red alert message indicating that the situation is critical.

2.3 Sizing considerations for 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, you need to consider caching costs. Cached pages in a Linux guest reduce the number of z/VM pages available to other z/VM guests.

- A large virtual memory space requires more 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 the performance of that guest. To reduce the penalty of occasional swapping that might occur in a smaller virtual machine, use fast swap devices, as discussed in Chapter 7, “Linux swapping” on page 103.

A 512 MB server does not require all of the memory, but will eventually appear to use all memory because its memory cost is four times that of the 128 MB server.

For more information about sizing practices for the Linux guest, see 5.5, “Sizing Linux virtual memory” on page 60.

Read more about the z/VM Performance Report at this link:

http://www.vm.ibm.com/perf/docs/

Additional z/VM performance tips are available at the following z/VM websites:

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

2.4 Benchmarking concepts and practices

Performance is the key metric in predicting when a processor is out of capacity. High processor utilization is desirable when considering ROI investment, but it is not a true predictor for capacity planning. As the workload increases on a processor, performance does not necessarily increase. If a processor is running at 90% utilization and workload grows by 10%, the performance will likely degrade. A system is out of capacity when the performance becomes unacceptable and does not improve with tuning.

A performance characteristic is scaling in a non-linear fashion relative to processor utilization. This means that response times can remain flat or nearly constant until utilization reaches a critical point, and then performance degrades and becomes erratic. This typically happens when you pass from the linear to the exponential part of the curve on a performance versus utilization chart. Also known as hitting the knee of the curve, this area is where performance can degrade quickly.

Every workload behaves differently at any given processor utilization. Work that is designated as high priority might perform well even when total utilization is at 100%. Low-priority work is the first workload impacted by an overloaded system. Performance for the low-priority work begins to elongate with slower response times and longer run times. Low-priority workload performance shows exponential degradation before high-priority work shows signs of stress. As the system nears maximum capacity, the workload can continue to grow, but system throughput might stay constant or decrease. Eventually the stress of an overloaded system affects all levels of workload (that is, low, medium, and high priorities).

Processor capacity and utilization are not the only factors affecting performance. Depending on the workload, performance is affected by all hardware components, including disk, memory, and network. The operating system, software subsystems, and application code all compete for those resources. The complexity of factors affecting performance means that there is no simple way to predict the effects of workloads and hardware changes. However, benchmarks remain a good way to evaluate the effects of changes to the computing environment.

Benchmarks differ from modeling in that real hardware, software, and applications are run instead of being simulated or modeled. A good benchmark consists of many iterations testing a workload, with changes made between iterations. The changes include one or more of these actions:

- Increase or decrease the workload.
- Increase, decrease, or change hardware resources.
- Adjust tuning parameters.

The benchmark team reviews the results of each iteration before determining the next adjustment. Benchmarks are an excellent opportunity to try many what-if scenarios. For more details on the benchmarks that we used for this book, see Appendix A, “WebSphere performance benchmark sample workload” on page 219.

3

z/VM and Linux monitoring tools

In this chapter, we analyze z/VM and Linux performance monitor data rather than discussing and comparing monitoring tools. However, we do include an overview of the tools used in the writing this of book as background information. It is important to understand the types of tools available for the System z environment and the value that they provide to a systems administrator, capacity planner, or performance specialist.

We begin by describing the performance data produced by the IBM z/VM and Linux performance monitoring software products. Beyond simple monitoring, these products provide charting capabilities, problem diagnosis and alerting, and integration with other software for monitoring of other systems such as z/OS®, databases, and web servers. The real-time monitoring tools used in writing this book include the VM Performance Toolkit and IBM Tivoli OMEGAMON XE for z/VM and Linux. The historical post-processing software products include the Tivoli Data Warehouse and IBM Tivoli OMEGAMON XE for z/VM and Linux. We continue to discuss performance tuning software products, and then conclude with simple tools or commands that are either provided with the operating systems or are available to the user community as free-ware.

Throughout this chapter, performance tuning tools are discussed in the context of the steps involved in analyzing system performance.

3.1 Introduction to system performance tuning tools

Sometimes the hardest part of improving system performance is getting started and selecting the tools. Many tool options are available:

- Real-time monitors and tools provide useful information
- Historical reporting tools provide health checks, problem determination, and trend analyses information.
- Sophisticated monitoring software can proactively check for performance issues and automatically act to prevent severe outages.

Real-time monitors can be as simple as Linux or z/VM commands built in each system. These commands can be a first option if more robust software products are not available. IBM software products such as the z/VM Performance Toolkit and Tivoli OMEGAMON XE on z/VM and Linux are more examples of real-time monitoring tools. We used them to capture performance information during our benchmarks. Throughout this book, the IBM Tivoli OMEGAMON suite of software is also referred to as OMEGAMON. The z/VM Performance Toolkit and OMEGAMON provide detailed system performance statistics in tailorable tabular and graphic formats. They also provide autonomic computing capabilities that respond to problem situations without operator intervention.

Historical reporting is outside the scope of most system commands. Consequently, health checks, problem determination, and trend analysis are best performed by software that collects the data for later analysis. After the data is collected, reporting tools can filter the massive amount of data, helping to narrow the search for relevant information. For this book, we choose the historical reporting tools IBM Tivoli OMEGAMON XE for z/VM and Linux, Tivoli Data Warehouse, and Tivoli Common Reporter.

3.2 IBM z/VM Performance Toolkit

The Performance Toolkit for VM, or the Performance Toolkit, is designed to assist operators, system programmers, and analysts in the following areas:

- System console operation in full panel mode

  The provided features facilitate VM systems operation, thus improving operator efficiency and productivity.

- Performance monitoring on z/VM systems

  An enhanced real-time performance monitor allows system programmers to monitor system performance and to analyze bottlenecks. Monitor features and data display improve the system programmer’s productivity when analyzing the system, allowing all new users to work efficiently. The Performance Toolkit helps system programmers to make more efficient use of system resources, increase system productivity, and improve user satisfaction.

The z/VM Performance Toolkit features include:

- Automatic threshold monitoring of many key performance indicators with operator notification if user-defined limits are exceeded.
- Special performance monitoring mode with displays for monitoring these:
  - General CPU performance
  - System and user storage utilization and management

  - Channel and I/O device performance, including cache data
  - Detailed I/O device performance, including information about the I/O load caused by specific minidisks on a real disk pack
  - General user data:
    - Resource consumption
    - Paging information
    - IUCV and VMCF communications
    - Wait states
    - Response times
  - Detailed user performance, including status and load of virtual devices
  - Summary and detailed information about shared file system servers
  - Configuration and performance information for TCP/IP servers
  - Linux performance data, for example, system execution space (SXS) performance data
  - Virtual networking configuration and performance data

  The collected performance data provides:

  - A redisplay facility for many of the key figures shown on the general CPU performance and storage utilization panels that allows browsing for up to 12 hours through the last measurements
  - Graphical history plots with a selection of up to four redisplay variables, either as simple plots, or, if the Graphical Data Display Manager program (GDDM®, 5684-007 or 5684-168) is available, also in the form of GDDM graphics
  - Graphical variable correlation plots, simple plots or GDDM graphics, that show how the values of specific variables are correlated to the values of another variable

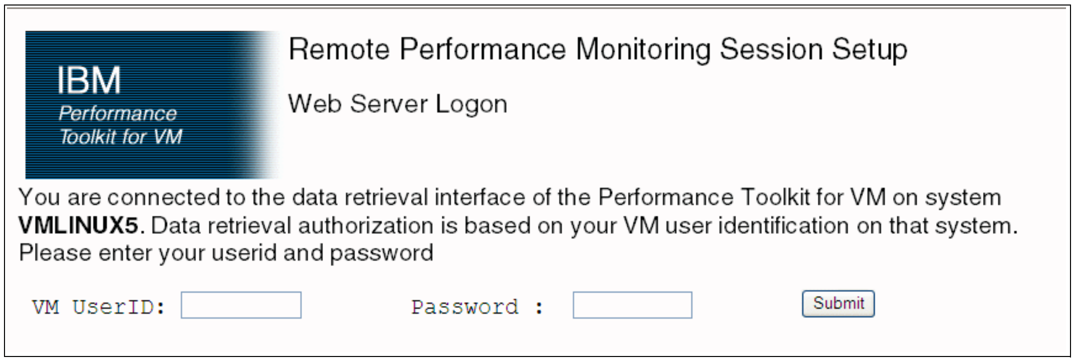
    Note: GDDM is required for generating graphics on the console of a virtual machine. However, no additional software is required when generating graphics with web browsers using the www interface.

- For later reference, the accumulated performance data can also be used for creating these:
  - Printed performance reports
  - Simple performance history files
  - Extended trend files
- For capacity planning, the history files on disk can be used for these:
  - Performance trend graphics
  - Historical data analysis

Over the past 17 years, the VM Performance Toolkit has evolved from an operations console with a few performance commands to a highly functional, varied coverage, deeply detailed reporting and real-time monitoring tool. It produces a consolidated view of the hardware, LPAR, VM, and Linux levels by communicating and collecting data from a Linux guest.

The z/VM Performance Toolkit provides detailed reports on all aspects of system and user performance information. The Performance Toolkit user interface is menu driven, but you can fastpath by entering the full or abbreviated report name to display a particular report. After a report is open, you can navigate further using sub menus, or by placing the cursor under a variable and pressing Enter. You can also drill down in many reports, for example, you can view an ordered list of DASD volumes along with overall activity and controller functions. Then you can select a volume for deeper details, including minidisk device activity. Figure 3-2 displays the menu.

This enhanced accessibility and navigation helps you to easily determine problems and improves your understanding of your system on a deep level. Data reports categories include general system data, I/O data, history data, and user data. To access a remote session of Performance Toolkit, use your installation IP address. You will be presented with the remote access web server logon (Figure 3-1). Enter your authorized user ID and password and click **Submit** to log on to the Toolkit.



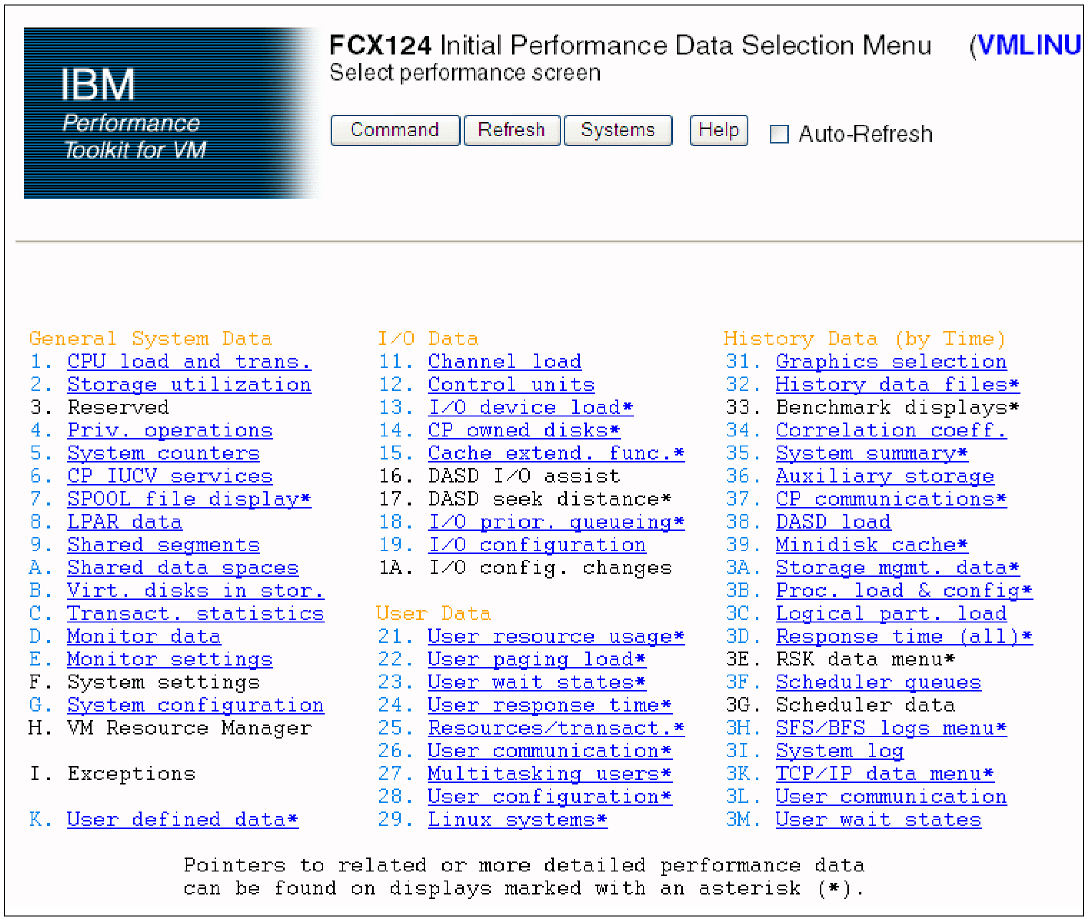
Remote Performance Monitoring Session Setup

Web Server Logon

You are connected to the data retrieval interface of the Performance Toolkit for VM on system **VMLINUX5**. Data retrieval authorization is based on your VM user identification on that system. Please enter your userid and password

VM UserID: Password : Submit

Figure 3-1 Performance Toolkit remote access



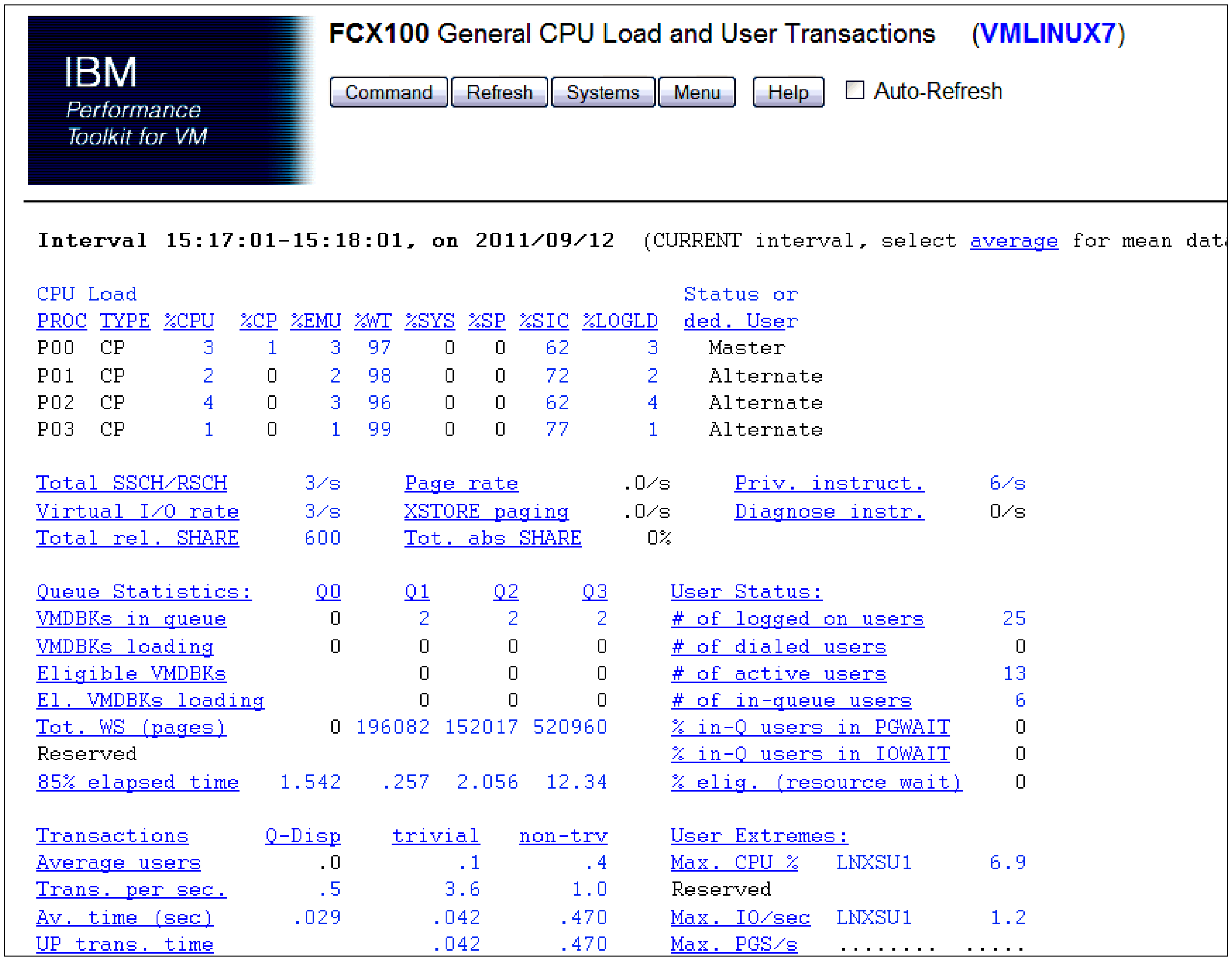
FCX124 Initial Performance Data Selection Menu (VMLINU

Select performance screen

```
General System Data          I/O Data                     History Data (by Time)
1. CPU load and trans.       11. Channel load             31. Graphics selection
2. Storage utilization       12. Control units            32. History data files*
3. Reserved                  13. I/O device load*         33. Benchmark displays*
4. Priv. operations          14. CP owned disks*          34. Correlation coeff.
5. System counters           15. Cache extend. func.*     35. System summary*
6. CP IUCV services          16. DASD I/O assist          36. Auxiliary storage
7. SPOOL file display*       17. DASD seek distance*      37. CP communications*
8. LPAR data                 18. I/O prior. queueing*     38. DASD load
9. Shared segments           19. I/O configuration        39. Minidisk cache*
A. Shared data spaces        1A. I/O config. changes      3A. Storage mgmt. data*
B. Virt. disks in stor.                                   3B. Proc. load & config*
C. Transact. statistics      User Data                    3C. Logical part. load
D. Monitor data              21. User resource usage*     3D. Response time (all)*
E. Monitor settings          22. User paging load*        3E. RSK data menu*
F. System settings           23. User wait states*        3F. Scheduler queues
G. System configuration      24. User response time*      3G. Scheduler data
H. VM Resource Manager       25. Resources/transact.*     3H. SFS/BFS logs menu*
                             26. User communication*      3I. System log
I. Exceptions                27. Multitasking users*      3K. TCP/IP data menu*
                             28. User configuration*      3L. User communication
K. User defined data*        29. Linux systems*           3M. User wait states

            Pointers to related or more detailed performance data
            can be found on displays marked with an asterisk (*).
```

Figure 3-2 z/VM Performance Toolkit menu

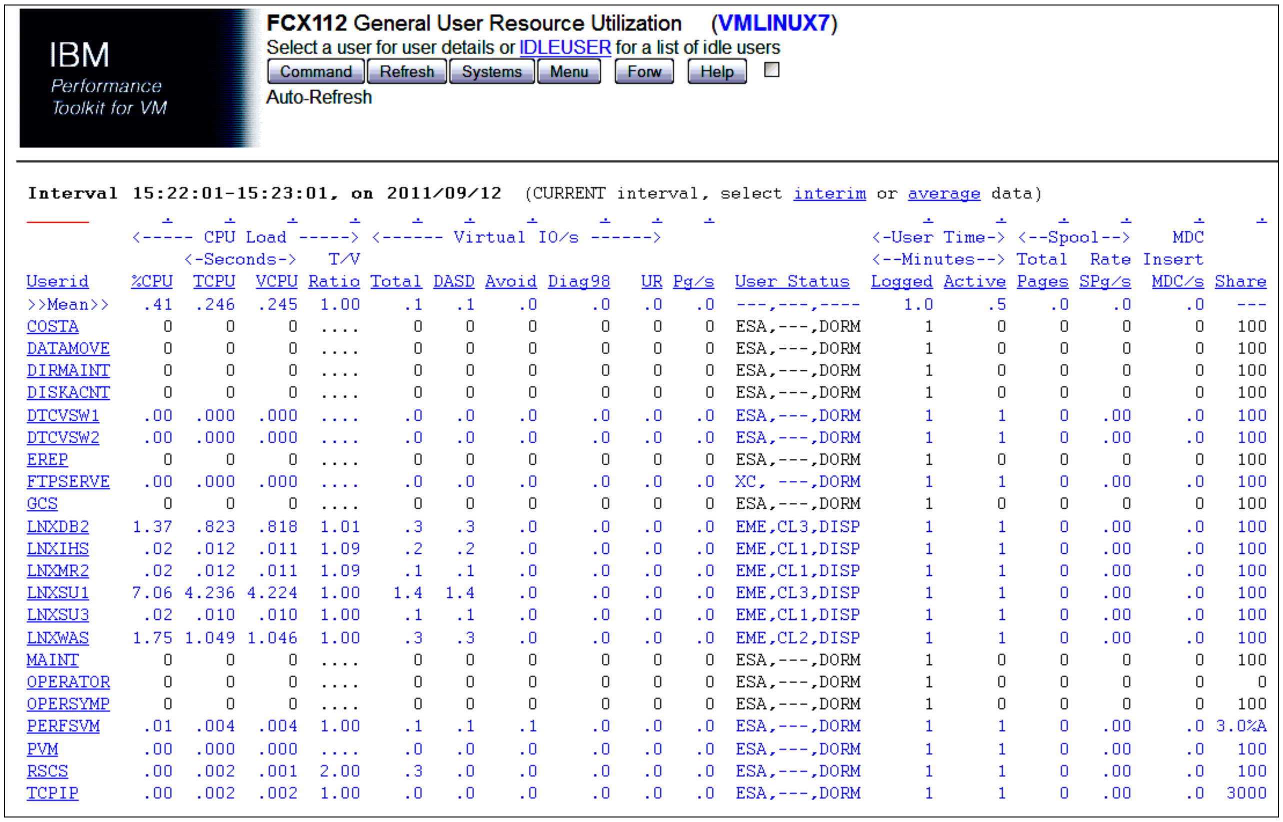
A good places to start are the categories of general system data and I/O data. Begin performance health check by selecting option **1. CPU load and trans**, which shows overall processor utilization, system contention, and information about the user population. The user data includes the total number of users, the number of active users, and the number of users who are waiting for a resource, for example, storage pages or I/O (Figure 3-3).



```
Interval 15:17:01-15:18:01, on 2011/09/12  (CURRENT interval, select average for mean dat

CPU Load                                                      Status or
PROC TYPE %CPU  %CP %EMU %WT %SYS %SP %SIC %LOGLD  ded. User
P00  CP      3    1    3  97    0   0   62      3    Master
P01  CP      2    0    2  98    0   0   72      2    Alternate
P02  CP      4    0    3  96    0   0   62      4    Alternate
P03  CP      1    0    1  99    0   0   77      1    Alternate

Total SSCH/RSCH      3/s   Page rate       .0/s   Priv. instruct.    6/s
Virtual I/O rate     3/s   XSTORE paging   .0/s   Diagnose instr.    0/s
Total rel. SHARE     600   Tot. abs SHARE    0%

Queue Statistics:     Q0     Q1     Q2     Q3    User Status:
VMDBKs in queue        0      2      2      2    # of logged on users       25
VMDBKs loading         0      0      0      0    # of dialed users           0
Eligible VMDBKs               0      0      0    # of active users          13
El. VMDBKs loading            0      0      0    # of in-queue users         6
Tot. WS (pages)        0 196082 152017 520960    % in-Q users in PGWAIT      0
Reserved                                         % in-Q users in IOWAIT      0
85% elapsed time   1.542   .257  2.056  12.34    % elig. (resource wait)     0

Transactions       Q-Disp    trivial   non-trv   User Extremes:
Average users          .0         .1        .4   Max. CPU %   LNXSU1      6.9
Trans. per sec.        .5        3.6       1.0   Reserved
Av. time (sec)       .029       .042      .470   Max. IO/sec  LNXSU1      1.2
UP trans. time                  .042      .470   Max. PGS/s   ........  .....
```

Figure 3-3 z/VM Performance Toolkit CPU display

Your next step depends on the results in the CPU display. The example shown in Figure 3-3 on page 17 is for a lightly loaded system with no performance issues. But if that display had shown high CPU utilization numbers, then your next step would be to view the main menu category of user data. The user data display (Figure 3-4) gives more details about which users are consuming the most processor resources. Select option 22 on the main menu for these results.



FCX112 General User Resource Utilization (VMLINUX7)

Select a user for user details or IDLEUSER for a list of idle users

```
Interval 15:22:01-15:23:01, on 2011/09/12  (CURRENT interval, select interim or average data)

          <----- CPU Load -----> <------ Virtual IO/s ------>                <-User Time-> <--Spool--> MDC
               <-Seconds->    T/V                                            <--Minutes--> Total  Rate Insert
Userid    %CPU  TCPU  VCPU Ratio Total DASD Avoid Diag98  UR Pg/s User Status Logged Active Pages SPg/s MDC/s Share
>>Mean>>   .41  .246  .245  1.00    .1   .1    .0     .0  .0   .0 ---,---,----   1.0     .5    .0    .0    .0   ---
COSTA        0     0     0  ....     0    0     0      0   0    0 ESA,---,DORM     1      0     0     0     0   100
DATAMOVE     0     0     0  ....     0    0     0      0   0    0 ESA,---,DORM     1      0     0     0     0   100
DIRMAINT     0     0     0  ....     0    0     0      0   0    0 ESA,---,DORM     1      0     0     0     0   100
DISKACNT     0     0     0  ....     0    0     0      0   0    0 ESA,---,DORM     1      0     0     0     0   100
DTCVSW1    .00  .000  .000  ....    .0   .0    .0     .0  .0   .0 ESA,---,DORM     1      1     0   .00    .0   100
DTCVSW2    .00  .000  .000  ....    .0   .0    .0     .0  .0   .0 ESA,---,DORM     1      1     0   .00    .0   100
EREP         0     0     0  ....     0    0     0      0   0    0 ESA,---,DORM     1      0     0     0     0   100
FTPSERVE   .00  .000  .000  ....    .0   .0    .0     .0  .0   .0 XC, ---,DORM     1      1     0   .00    .0   100
GCS          0     0     0  ....     0    0     0      0   0    0 ESA,---,DORM     1      0     0     0     0   100
LNXDB2    1.37  .823  .818  1.01    .3   .3    .0     .0  .0   .0 EME,CL3,DISP     1      1     0   .00    .0   100
LNXIHS     .02  .012  .011  1.09    .2   .2    .0     .0  .0   .0 EME,CL1,DISP     1      1     0   .00    .0   100
LNXMR2     .02  .012  .011  1.09    .1   .1    .0     .0  .0   .0 EME,CL1,DISP     1      1     0   .00    .0   100
LNXSU1    7.06 4.236 4.224  1.00   1.4  1.4    .0     .0  .0   .0 EME,CL3,DISP     1      1     0   .00    .0   100
LNXSU3     .02  .010  .010  1.00    .1   .1    .0     .0  .0   .0 EME,CL1,DISP     1      1     0   .00    .0   100
LNXWAS    1.75 1.049 1.046  1.00    .3   .3    .0     .0  .0   .0 EME,CL2,DISP     1      1     0   .00    .0   100
MAINT        0     0     0  ....     0    0     0      0   0    0 ESA,---,DORM     1      0     0     0     0   100
OPERATOR     0     0     0  ....     0    0     0      0   0    0 ESA,---,DORM     1      0     0     0     0     0
OPERSYMP     0     0     0  ....     0    0     0      0   0    0 ESA,---,DORM     1      0     0     0     0   100
PERFSVM    .01  .004  .004  1.00    .1   .1    .1     .0  .0   .0 ESA,---,DORM     1      1     0   .00    .0 3.0%A
PVM        .00  .000  .000  ....    .0   .0    .0     .0  .0   .0 ESA,---,DORM     1      1     0   .00    .0   100
RSCS       .00  .002  .001  2.00    .3   .0    .0     .0  .0   .0 ESA,---,DORM     1      1     0   .00    .0   100
TCPIP      .00  .002  .002  1.00    .0   .0    .0     .0  .0   .0 ESA,---,DORM     1      1     0   .00    .0  3000
```

Figure 3-4 z/VM Performance Toolkit User Resource Util display

For more details about using the Performance Toolkit to determine the system, see *Linux on IBM eServer zSeries and S/390: Performance Toolkit for VM*, SG24-6059.

3.3 IBM Tivoli OMEGAMON XE on z/VM and Linux

Tivoli OMEGAMON XE on z/VM and Linux is one of a suite of Tivoli Management Services products. These products use common shared technology components to monitor your mainframe and distributed systems on a variety of platforms and provide workstation-based reports that you can use to track trends and to understand and troubleshoot system problems.

Tivoli OMEGAMON XE on z/VM and Linux V4.2.0 is a powerful product that displays data collected from multiple systems on one easy-to-use, flexible interface. With this monitoring agent, you can view z/VM data obtained from the Performance Toolkit and performance data from Linux on System z. This dual capability allows you to solve problems quickly and helps you manage a complex environment.

OMEGAMON monitoring software provides a portal that is accessible via a desktop client or a browser. The portal interface allows you to monitor systems without logging into either Linux or z/VM. The portal provides a single point of control for monitoring all of Linux and z/VM images, in addition to z/OS and its subsystems, databases, web servers, network services, and WebSphere MQ.

The OMEGAMON portal displays the following types of z/M and Linux data:

- z/VM Memory usage, disk usage, and paging activity by workload name
- z/VM System-wide data by resource, such as logical partition usage and paging data
- z/VM System-wide spooling space usage
- z/VM TCP/IP network statistics for enabled network server virtual machines
- z/VM HiperSockets™ utilization data
- z/VM CPU data
- Linux on IBM System z workload statistics
- Linux disk utilization
- Linux file usage
- Linux process resource usage
- Linux system
- Linux user resource usage

3.3.1 Reporting on historical data using OMEGAMON

In addition to seeing real-time data in table and chart views, you can use Tivoli Data Warehouse to store the data and examine it over time. The table view and the bar, pie, and plot charts have a time span setting tool that displays data samples reported up to the time specified.

OMEGAMON XE on z/VM and Linux contains workspaces that provide additional views of historical data. The historical views can be used to graph or report on data from previous days, weeks, months, and so on.

3.3.2 Problem determination using OMEGAMON

To being checking system performance using OMEGAMON, examine the high-level displays for each system. In our environment, we are concerned only with the performance of z/VM and its Linux guests. The z/VM OMEGAMON display is a good starting point because z/VM controls all of the hardware resources and virtualizes that environment for Linux guests.

Beginning with Figure 3-5, the following figures highlight the key performance indicators of CPU, disk, and storage utilizations. You can view system utilization data in graph format, as shown in the following figures, or in text or table format by selecting the appropriate option on the OMEGAMON portal.

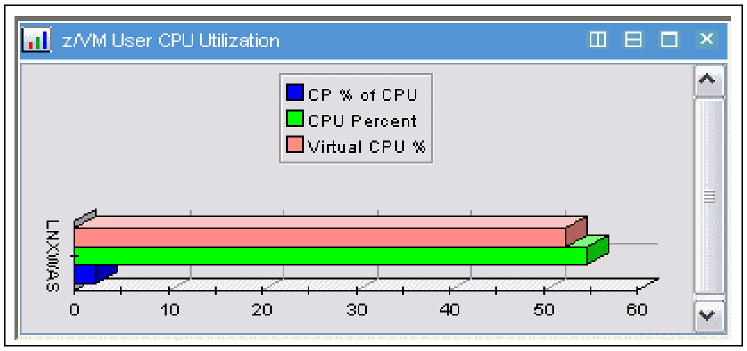


Figure 3-5 OMEGAMON CPU utilization graph

Figure 3-6 displays system device utilization in graph form.

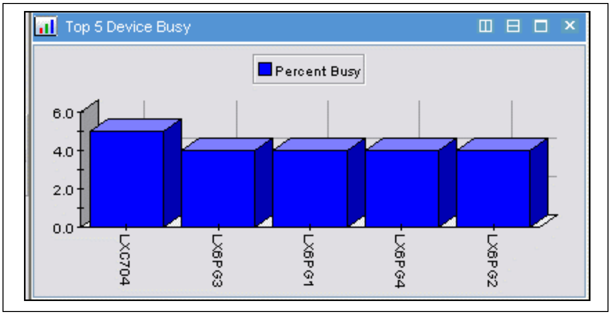


Figure 3-6 OMEGAMON device utilization graph

Figure 3-7 displays system storage utilization in graph form.

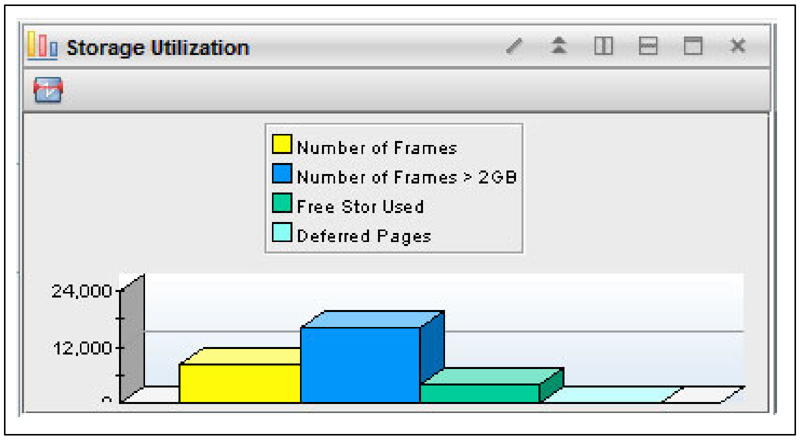
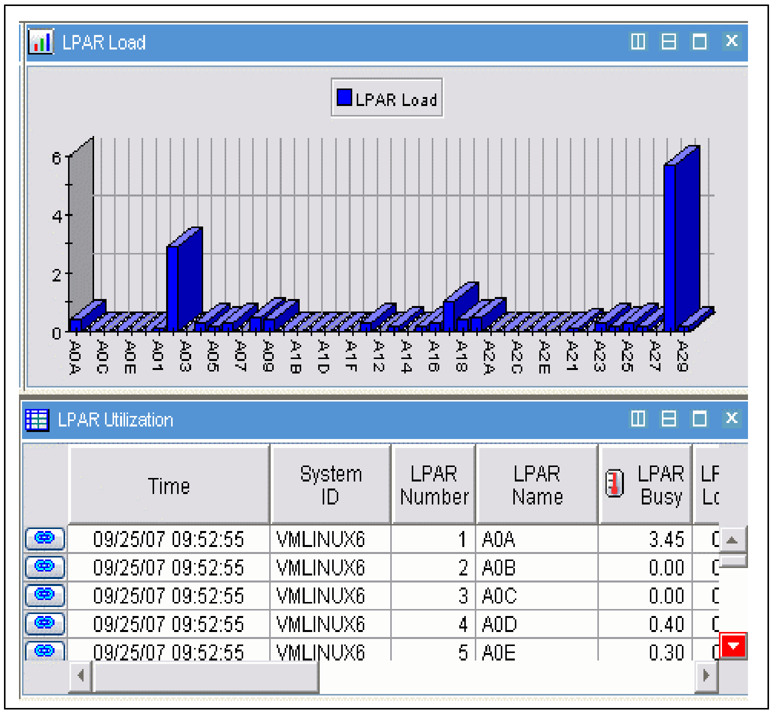


Figure 3-7 OMEGAMON storage utilization graph

You can share processors in a System z environment between LPARS, which means that processors assigned to this z/V can be shared by other z/VM, Linux, or z/OS systems. This requires that you examine the total hardware complex in your system as you begin problem solving. Figure 3-8 displays this type of data with a bar chart showing processor utilization of every LPAR on the System z plus a table with detailed information about each LPAR. Use the scroll bars to expand the table to show additional columns and rows. Click the icon displayed on the left of each row to view row or LPAR details.



Time	System ID	LPAR Number	LPAR Name	LPAR Busy	LF Lo
09/25/07 09:52:55	VMLINUX6	1	A0A	3.45	C
09/25/07 09:52:55	VMLINUX6	2	A0B	0.00	C
09/25/07 09:52:55	VMLINUX6	3	A0C	0.00	C
09/25/07 09:52:55	VMLINUX6	4	A0D	0.40	C
09/25/07 09:52:55	VMLINUX6	5	A0E	0.30	C

Figure 3-8 OMEGAMON LPAR utilization graph

At this point, your next step in the performance evaluation is to examine statistics for the Linux guests running on z/VM. The overview workspace for Linux guests displays many key performance indicators. Figure 3-9 displays top Linux systems and their CPU utilization, their load on the system for the last 15 minutes, and a table of system statistics such as context switches, paging rates, length of time that the system has been up, and the number of logged-on users.

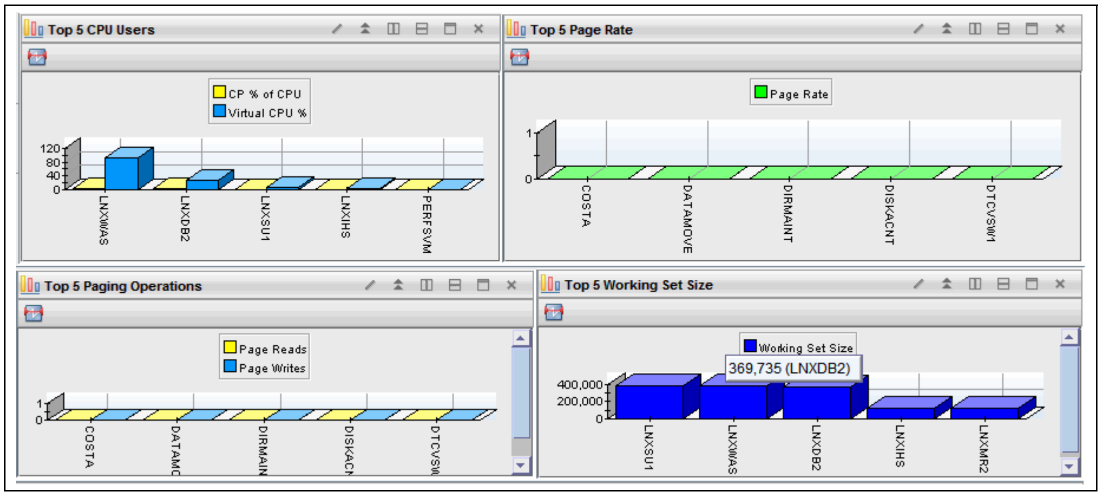


Figure 3-9 OMEGAMON Linux guest performance overall view

Figure 3-10 displays system details for the top five Linux guests.

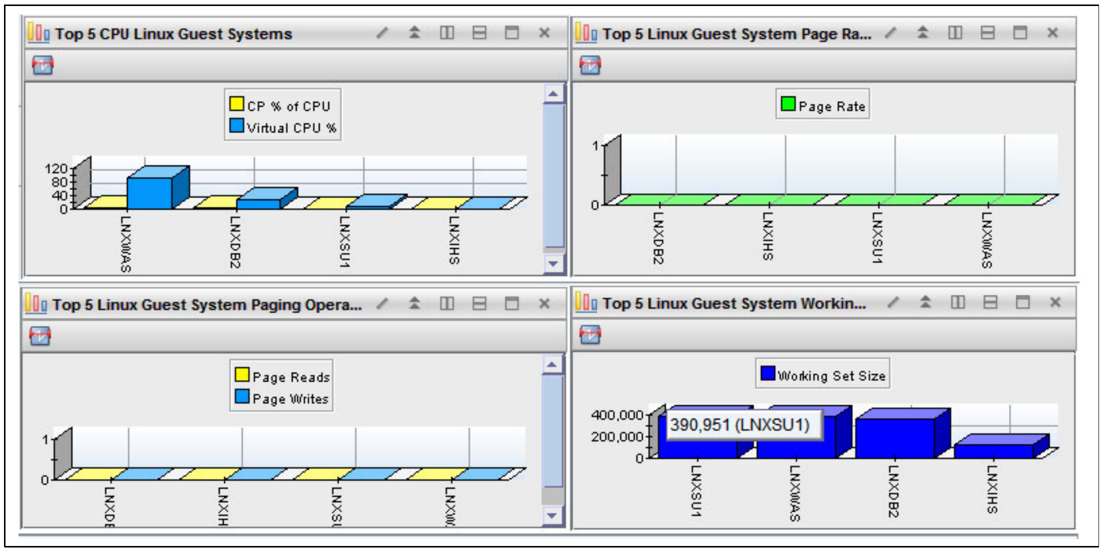
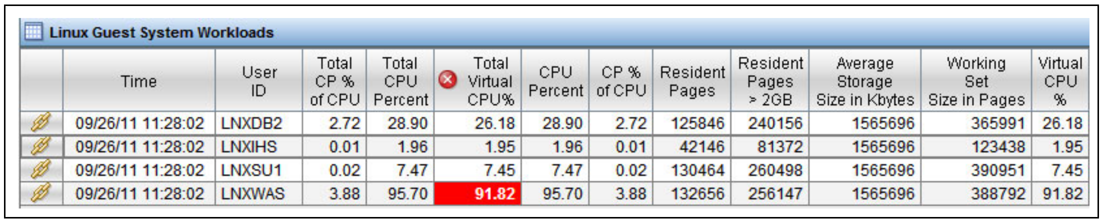


Figure 3-10 OMEGAMON Linux system graph

Figure 3-11 displays workload details for Linux guest systems.

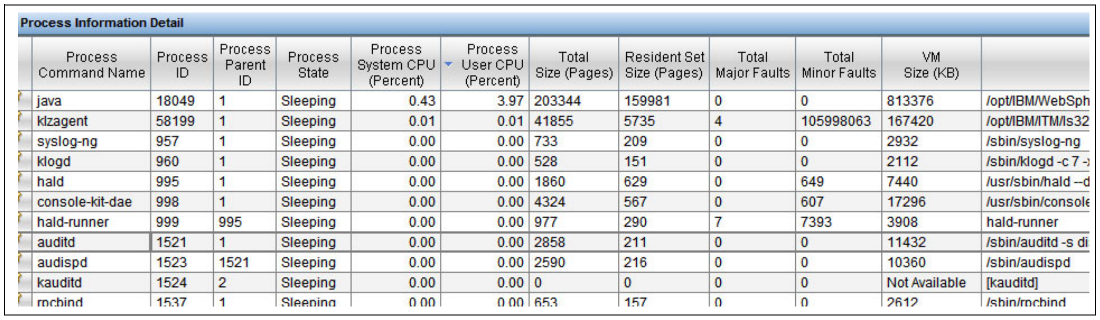


Linux Guest System Workloads

Time	User ID	Total CP % of CPU	Total CPU Percent	Total Virtual CPU%	CPU Percent	CP % of CPU	Resident Pages	Resident Pages > 2GB	Average Storage Size in Kbytes	Working Set Size in Pages	Virtual CPU %
09/26/11 11:28:02	LNXDB2	2.72	28.90	26.18	28.90	2.72	125846	240156	1565696	365991	26.18
09/26/11 11:28:02	LNXIHS	0.01	1.96	1.95	1.96	0.01	42146	81372	1565696	123438	1.95
09/26/11 11:28:02	LNXSU1	0.02	7.47	7.45	7.47	0.02	130464	260498	1565696	390951	7.45
09/26/11 11:28:02	LNXWAS	3.88	95.70	91.82	95.70	3.88	132656	256147	1565696	388792	91.82

Figure 3-11 OMEGMON Linux application table

Figure 3-12 shows the status and resource utilization for each process. You can sort and filter table data.



Process Information Detail

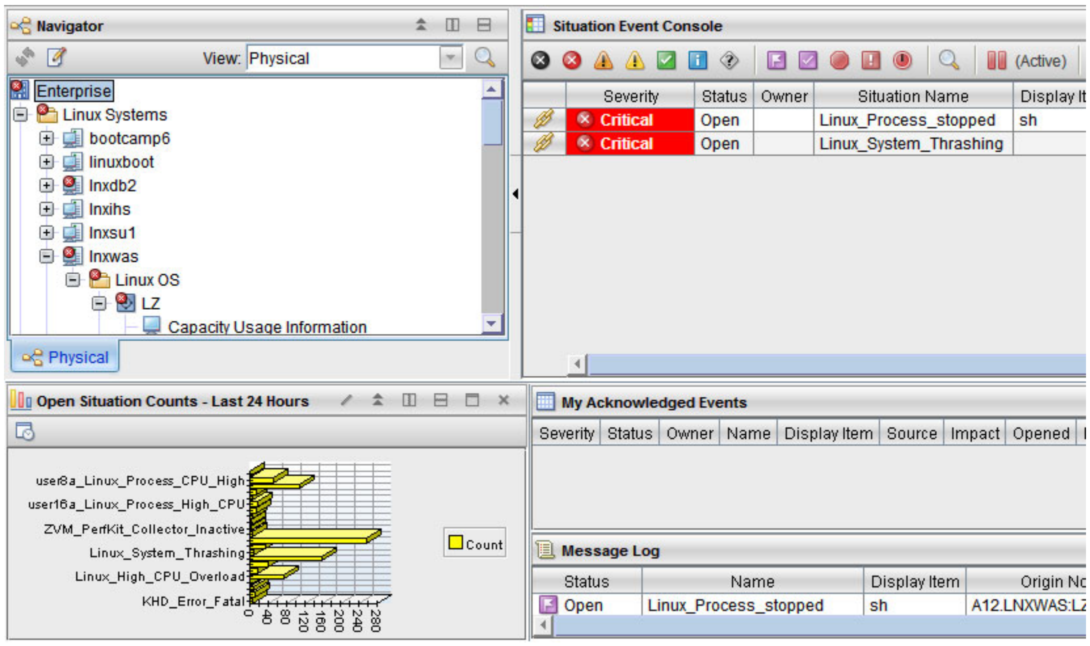
Process Command Name	Process ID	Process Parent ID	Process State	Process System CPU (Percent)	Process User CPU (Percent)	Total Size (Pages)	Resident Set Size (Pages)	Total Major Faults	Total Minor Faults	VM Size (KB)	
java	18049	1	Sleeping	0.43	3.97	203344	159981	0	0	813376	/opt/IBM/WebSph
klzagent	58199	1	Sleeping	0.01	0.01	41855	5735	4	105998063	167420	/opt/IBM/ITM/ls32
syslog-ng	957	1	Sleeping	0.00	0.00	733	209	0	0	2932	/sbin/syslog-ng
klogd	960	1	Sleeping	0.00	0.00	528	151	0	0	2112	/sbin/klogd -c 7 -
hald	995	1	Sleeping	0.00	0.00	1860	629	0	649	7440	/usr/sbin/hald --d
console-kit-dae	998	1	Sleeping	0.00	0.00	4324	567	0	607	17296	/usr/sbin/console
hald-runner	999	995	Sleeping	0.00	0.00	977	290	7	7393	3908	hald-runner
auditd	1521	1	Sleeping	0.00	0.00	2858	211	0	0	11432	/sbin/auditd -s di
audispd	1523	1521	Sleeping	0.00	0.00	2590	216	0	0	10360	/sbin/audispd
kauditd	1524	2	Sleeping	0.00	0.00	0	0	0	0	Not Available	[kauditd]
rpcbind	1537	1	Sleeping	0.00	0.00	653	157	0	0	2612	/sbin/rpcbind

Figure 3-12 OMEGAMON Linux processes table

You can perform these tasks using the Tivoli OMEGAMON XE for z/VM and Linux:

- Navigate to deeper levels of performance data details. For Linux guests, this monitoring agent provides links to Tivoli Monitoring Agent for Linux OS workspaces directly from the Tivoli OMEGAMON XE on z/VM and Linux workspaces.
- Connect to the z/VM host system via TCP/IP to access the Performance Toolkit panels.
- Read expert advice about how to identify and resolve performance problems.

IBM Tivoli OMEGAMON comes with default thresholds for key system performance indicators. When a threshold is exceeded, OMEGAMON generates a situation alert and takes an automated action to address the problem. The situation console log workspace shows the active situation alerts. In Figure 3-13, the upper panel lists open situations. The lower left panel has the situation count for the last 24 hours. The lower right panel lists the situation events that the user has already acknowledged.



Severity	Status	Owner	Situation Name	Display It
Critical	Open		Linux_Process_stopped	sh
Critical	Open		Linux_System_Thrashing	

Severity	Status	Owner	Name	Display Item	Source	Impact	Opened

Status	Name	Display Item	Origin No
Open	Linux_Process_stopped	sh	A12.LNXWAS:LZ

Figure 3-13 OMEGAMON situation alerts

Default workspaces and situations allow you to monitor your enterprise as soon as the Tivoli OMEGAMON XE on z/VM and Linux software is installed and configured. The user interface offers several formats to view data including graphs, bar charts, and tables. You can customized workspaces and situations to suit your unique business needs.

3.3.3 IBM Tivoli OMEGAMON monitoring options for Linux systems

When you choose monitoring options for a z/VM environment with Linux guests, remember to consider the types and quantity of monitoring data that you need to collect in addition to system overhead. A typical z/VM and Linux production environment consists of several z/VM systems with perhaps hundreds of Linux guests. Some Linux systems run mission-critical work and must be monitored at a detail level via agents installed on each guest. Other Linux systems run less critical workloads and can be monitored by Simple Network Management Protocol (SNMP) monitoring agents. Finally, certain Linux guests might need no monitoring. IBM Tivoli OMEGAMON XE for z/VM and Linux offers both agent and SNMP monitoring options for Linux systems. We discuss various monitoring options in the following sections.

Agentless SNMP monitoring on Linux

IBM Tivoli Monitoring is the base software for the Agentless SNMP Monitor for Linux. IBM Tivoli Monitoring monitors the availability and performance of all the systems in your enterprise from one or more designated workstations. It also provides useful historical data that you can use to track trends and to troubleshoot system problems. You can use IBM Tivoli Monitoring to perform these tasks:

- Monitor for alerts on the system with predefined situations or custom situations.
- Establish your own performance thresholds.
- Trace the causes leading to an alert.
- Gather comprehensive data about system conditions.
- Use policies to perform actions, schedule work, and automate manual tasks.

The Agentless Monitor for Linux software uses standard SNMP to identify common problems and to notify you when they occur. SNMP software includes the following features:

- Monitoring
- Data gathering
- Event management

The Agentless Monitor for Linux communicates with the systems or subsystems that you want to monitor. A single monitor can collect data from a large number of Linux systems, depending on their level of activity. It collects and distributes this data to a Tivoli Enterprise Portal Server.

Monitoring with ITM Linux OS agent

The IBM Tivoli Monitoring Agent for Linux OS sends performance and event data to the Tivoli Enterprise Management Server similar to the agentless SNMP method. However, it requires installing an agent on each system that will be monitored. This agent provides more extensive data collection and event management capabilities than SNMP.

As part of the Tivoli Enterprise Portal for Distributed Systems, the Monitoring Agent for Linux OS offers a central management point for Linux environments and provides a comprehensive means for gathering information to detect and prevent problems. You can monitor multiple servers from a single workstation, and information is standardized across the entire system.

The Monitoring Agent for Linux OS is an intelligent monitoring agent that resides on managed resources. It helps to predict problems and sends alerts when critical events occur. Systems administrators can set threshold levels and receive alerts when thresholds are reached.

IBM Tivoli monitors provide data displays called workspaces that use large groups of data attributes to show event and performance statistics for z/VM and Linux systems. Both offer product-provided alerting.

We used IBM Tivoli OMEGAMON XE for z/VM and Linux to monitor and analyze the test environment created for this book. OMEGAMON offers the options of using both installed agents for collecting detailed monitoring data, and remote agents using SNMP to monitor lower-priority Linux systems. SNMP monitoring has the lowest overhead but is limited in the types of performance data that can be collected.

An installed IBM Tivoli agent has an overhead of approximately 0.01 - 0.03% of a processor for each guest. Even with the minimal overhead, an agent on every Linux guest might not be necessary. You can use SNMP monitors, or remote monitors, for Linux systems that do not need high levels of monitoring and performance tuning.

3.4 z/VM CP system commands

Similar to Linux, the z/VM operating system has control program (CP) commands to display system performance information. You do not need to include the CP command qualifier when you type commands. z/VM responds to either **`cp indicate`** or **`indicate`** or **`ind`**.

The following section describes the major CP commands used in gathering performance data.

Note: You need class E authority to be defined in your userid directory.

3.4.1 cp indicate

The **`cp indicate`** command results in a snapshot of resource utilization (Figure 3-14). It displays information about overall processor utilization, or LPAR processor utilization, paging rate, mini-disk cache rate, and user queues. The command displays results based on the user's class - B, C, E, or G. **`cp indicate`** is a good starting point in analyzing system performance issues. The default option of this command is LOAD.

```
cp indicate
AVGPROC-003% 04
XSTORE-000000/SEC MIGRATE-0000/SEC
MDC READS-000001/SEC WRITES-000000/SEC HIT RATIO-100%
PAGING-0/SEC STEAL-000%
Q0-00000(00000)                           DORMANT-00018
Q1-00001(00000)           E1-00000(00000)
Q2-00001(00000) EXPAN-001 E2-00000(00000)
Q3-00003(00000) EXPAN-001 E3-00000(00000)

PROC 0000-004% CP       PROC 0001-003% CP
PROC 0002-004% CP       PROC 0003-003% CP

LIMITED-00000
```

Figure 3-14 VM command - cp indicate

In Figure 3-14:

- AVGPROC indicates utilization of all processors.
- MIGRATE indicates whether working sets of the logged-on virtual machines fit in central and expanded storage. A growing MIGRATE number indicates that central and expanded storage are insufficient to contain all working sets. The higher the migrate number, the greater your workload's paging demands are on the system.
- STORAGE indicates the real storage utilization.
- Q0 - Q3 indicates virtual machines in the dispatch list.
- E1 - E3 indicates virtual machines in the eligible list.
- DORMANT indicates virtual machines in the dormant list.

3.4.2 cp indicate active

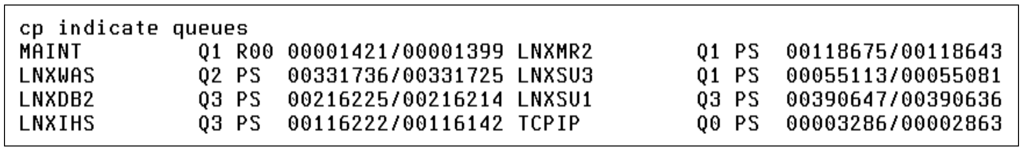
This command displays the total number of active users for a specific time. It also displays numbers of users in the dispatch, elegible, and dormant lists that were active in a certain period of time.

3.4.3 cp indicate queues

This command displays user IDs and their associated dispatching queue. Users who have a virtual multiprocessor will show multiple entries for a single user. Figure 3-15 displays an example of indicate queues.

In Figure 3-15, the first column is the list of virtual machines in priority order. The second column shows the users list. If this column frequently shows users in the E1, E2, or E3 list, you might have a constraint in system resources. The third column explains why a virtual machine is in a wait state:

- Rnn: Current RUNUSER on the specified real processor, where *nn* is the processor ID
- IO: Waiting for I/O
- PS: PSW wait (enabled wait state)
- PG: Waiting for paging



```
cp indicate queues
MAINT        Q1 R00 00001421/00001399 LNXMR2       Q1 PS  00118675/00118643
LNXWAS       Q2 PS  00331736/00331725 LNXSU3       Q1 PS  00055113/00055081
LNXDB2       Q3 PS  00216225/00216214 LNXSU1       Q3 PS  00390647/00390636
LNXIHS       Q3 PS  00116222/00116142 TCPIP        Q0 PS  00003286/00002863
```

Figure 3-15 VM command - Indicate queues

3.4.4 cp indicate i/o

`indicate I/O` identifies virtual machines currently in an I/O wait state along with the real I/O device number that they are waiting for. You can repeat this command to see a pattern.

Indicate I/O shows virtual machines currently in an I/O wait state and also shows the real device number to which the most recent I/O operation was mapped. Four dashes (----) indicates a virtual device.

If you see the same real device number for several virtual machines, there are too many minidisks on the same DASD or you have a DASD controller bottleneck where too many DASD are doing I/O on the same controller.

3.4.5 cp indicate user

`indicate user` displays the resources used, or occupied, by a virtual machine or by the system. CP displays a set of statistics for each virtual machine. If you do not type a user ID after the command, the default is the user ID issuing the command.

The command indicate user shows these:

- The virtual machine definition as USERID, STOR, and other characteristics
- The number of pages in real and expanded storage

To check whether this user ID is running, issue this command several times to display the amount of resources changing or the number of I/Os increasing.

3.4.6 cp indicate paging

This command displays a list of the virtual machines in page wait status. It also displays the number of pages that this user ID has in the expanded storage and in the auxiliary storage (DASD).

For more details about the z/VM **indicate** commands, see:

http://www.redbooks.ibm.com/abstracts/tips0592.html?Open

3.4.7 CP Query commands

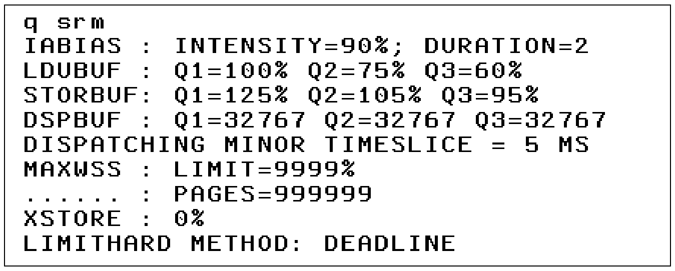
CP Query commands are another set of z/VM commands that display system and user information.

cp query srm

cp query srm displays the system-wide parameters that the scheduler uses to set the priority of system resource access. These parameters define the size of the time slice and the access to resources for different user classes. Each of these parameters has a default setting that is appropriate for most environments. Use the **cp set srm** command if you need to control use of resources dependent on user class. Note that eligible lists can occur if this command is not used carefully. We discuss adjusting default settings later in this book.

System parameters displayed from **cp query srm** include these (Figure 3-16).

- iabias: Interactive bias used to favor short transactions
- ldubuf: Controls z/VM's tolerance for guests that induce paging
- storbuf: Used to encourage z/VM to overcommit main memory
- Dispatching minor timeslice: Controls the length of time that a single user holds onto a processor



```
q srm
IABIAS : INTENSITY=90%; DURATION=2
LDUBUF : Q1=100% Q2=75% Q3=60%
STORBUF: Q1=125% Q2=105% Q3=95%
DSPBUF : Q1=32767 Q2=32767 Q3=32767
DISPATCHING MINOR TIMESLICE = 5 MS
MAXWSS : LIMIT=9999%
...... : PAGES=999999
XSTORE : 0%
LIMITHARD METHOD: DEADLINE
```

Figure 3-16 VM command - cp query srm

cp query allocate page

cp query alloc page displays detailed information about the paging space, including the number of cylinders or pages that are allocated, in use, and available for DASD volumes attached to the system. The **q alloc spool** command displays similar details on spool space. You can define it dynamically, but this definition will be lost if the system restarts. To define the definition permanently, define it in the System Config file in the PARM disk of z/VM.

cp query alloc page displays a list of the paging DASDs and the allocation amount at that specific moment, and also shows a summary of the total amount of usage.

Note: Be careful to not mix different types of DASD or different models, as that might affect the algorithms and the system performance.

cp query share

Use this command, with the user ID, to display the percentage of the system available resources that a user ID is allowed to use. This can be set as *absolute* or *relative*, with the default being a relative percentage of 100 for each user ID.

Note: A virtual machine receives the portion of resources (that is, processors, real storage, and so on) defined in its SHARE setting.

cp query quickdsp

Use this command, with the user ID, to display whether this option is on or off for that specific user ID. If it is on, the user ID will be put in the eligible list without waiting on the queues. For more details, see "CP set quickdsp command" on page 30.

Table 3-1 describes additional query commands that you can use when investigating system performance issues.

Table 3-1 Some CP QUERY commands

Query	Description
CACHE	QUERY CACHE displays the caching status for all storage subsystems that support caching. Use this to investigate I/O-related problems.
CHPIDS	QUERY CHPIDS displays all 256 machine channel paths and their physical status. I/O performance problems can occur if CHPIDs are offline or are not available.
CPLOAD	QUERY CPLOAD displays information about the last CP IPL, including location of the last used CP module, the parm disk location, and how CP was started.
CPOWNED	QUERY CPOWNED displays the list of CP-owned DASD volumes. Check this list when paging or spooling problems occur to make sure that all required volumes are available.
FRAMES	QUERY FRAMES displays the status of host real storage. Use this when users are shown in the eligible list.
MDC	Use this to investigate I/O problems. QUERY MDC from a Class B will: ► Query minidisk cache settings for the entire system, a real device, an active minidisk, or a minidisk defined in the directory. ► Query a user's ability to insert data into the cache.
PATHS	QUERY PATHS displays all paths installed to a specific device or range of devices and installed path status. Use this to investigate I/O problems.
PENDING	QUERY PENDING displays the entered device commands for which the associated asynchronous function has not been completed. Use this to investigate hung users or I/O problems.
QIOASSIST	QUERY QIOASSIST determines the current status of the queue-I/O assist for a virtual machine. Use this to investigate I/O or networking problems.
RESERVED	QUERY RESERVED displays the number of reserved real storage frames. Use SET RESERVED to reserve pages of storage for a user. This is good for tuning users in a storage-constrained system.
SRM	QUERY SRM displays the settings of the System Resource Manager, including IABIAS, LDUBUF, and STORBUF.
STOR	Use QUERY STORAGE or QUERY STORE to display the size of real storage.
SYSTEM	QUERY SYSTEM displays current user access to a system DASD volume.
XSTOR	QUERY XSTORAGE or QUERY XSTORE displays the assignment of real Expanded Storage. Use this to investigate response time or paging problems.

3.4.8 CP Set commands

CP Set commands change performance characteristics for both the entire system and a single user.

cp set share command

The **`cp set share`** command changes the system-resource-access priority for users. There are two parameters:

- ABSolute *nnn*%

  This specifies that this user will receive a target minimum of *nnn*% of the scheduled system resources including CPU, storage, and paging capacity.

- RELative nnnnn

  This specifies that this user will receive a target minimum relative share of nnnnn. The amount of scheduled system resources available to relative share users is the total of resources available, less the amount allocated to absolute share users.

cp set srm command

The **`cp set srm command`** sets some z/VM system tuning parameters. These parameters define the size of the time slice, the access to the resources for different user classes as seen by the scheduler. Each of these parameters has a default setting that is appropriate for most environments. We discuss adjustments to the defaults later in this book. The System Resource Manager (srm) parameters contained in this display include these:

- iabias: Interactive bias, used to favor short transactions
- ldubuf: Controls z/VM's tolerance for guests that induce paging
- storbuf: Can be used to encourage z/VM to overcommit main memory
- dispatching minor timeslice: Controls the length of time that a single user holds onto a processor
- maxwss
- limithard: Sets the enforcement of hard limiting of scheduled system resources. This setting only affects users with absolute maximum shares defined with the SET SHARE command or the SHARE directory statement.

CP set quickdsp command

When you use this command on a virtual machine, that virtual machine is added to the dispatch list immediately without waiting in the eligible list. It goes into the special queue Q0, where it is dispatched as soon as possible, instead of normal queues like Q1, Q2, or Q3. Figure 3-17 shows the command results.

```
set quickdsp mntlinux on
USER MNTLINUX:  QUICKDSP = ON
Ready; T=0.01/0.01 14:24:04
```

Figure 3-17 QUICKDSP command

Table 3-2 describes additional CP SET commands.

Table 3-2 CP SET commands

Set	Description
MDC	Use this when there is not much minidisk activity and you need to release storage. SET MDC from a Class B will: ► Change minidisk cache settings for the entire system, for a real device, or for an active minidisk. ► Purge the cache of data from a real device or an active minidisk. ► Change a user's ability to insert data into the cache.
QIOASSIST	SET QIOASSIST controls the queue-I/O assist (QDIO performance assist for V=V guests) for a virtual machine. This interpretive-execution assist applies to devices that use the Queued Direct I/O (QDIO) architecture, HiperSockets devices, and FCP devices.
RESERVED	SET RESERVED establishes the number of real storage frames that are available to a specific virtual machine. Use this to tune specific guests in a storage constrained environment.

For further details on command syntax, see the *z/VM Help information or z/VM CP Command and Utilities Reference*, SC24-6175.

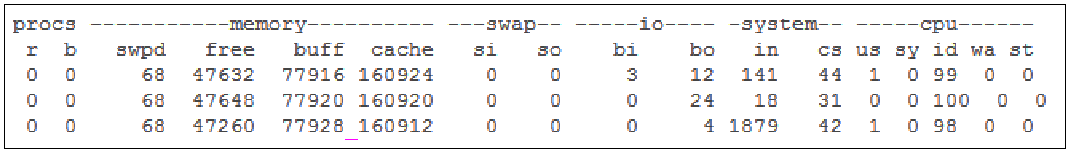
3.5 Linux system tools

Use Linux system simple commands as the initial step in evaluating system performance. These commands are either standard to Linux or available as free-ware on the internet.

3.5.1 vmstat command

The **vmstat** command displays current statistics for processes, usage of real and virtual memory, paging, block I/O, and CPU.

Figure 3-18 displays data and adds an update line three times, at five-second intervals. The left-most columns show the number of running processes and number of blocked processes. The memory section shows memory being swapped out, free memory, the amount of buffer containing inodes and file metadata, and cached memory for files being read from disk. The swap section lists swaps in and swaps out. The I/O section reports the number (kb) of blocks read in and written out. System in and cs represent interrupts and context switches per second. The CPU section headers of us, sy, id, and wa represent the percentage of CPU time count on users, system, idle, I/O wait, and steal, respectively. Use **vmstat** to identify memory shortages and I/O wait issues, and in situations where virtual CPUs are not backed up by physical CPUs.

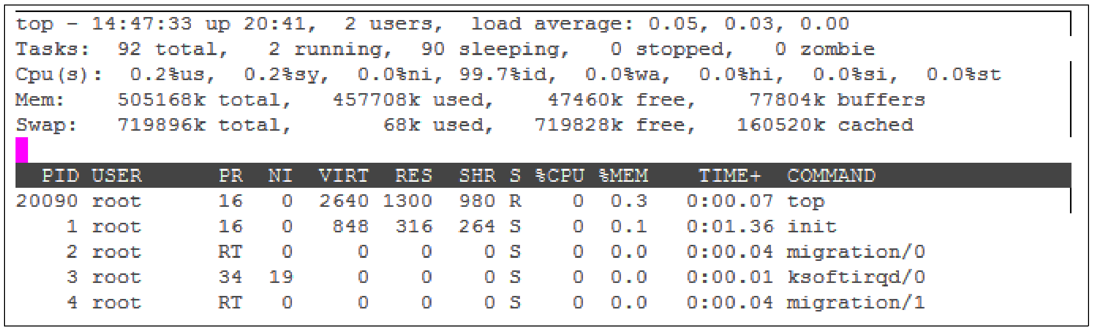


```
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0     68  47632  77916 160924    0    0     3    12  141   44  1  0 99  0  0
 0  0     68  47648  77920 160920    0    0     0    24   18   31  0  0 100  0  0
 0  0     68  47260  77928 160912    0    0     0     4 1879   42  1  0 98  0  0
```

Figure 3-18 Linux command - vmstat

3.5.2 Top command

The **top** command displays process statistics and a list of the top Linux processes using the CPU. You can interactively customize the format and content of the display. Sort the process list run time, memory usage, and processor usage. **top** shows the total number of processes, the number running, and the number sleeping. Information is updated every three seconds, or at a chosen interval. Use this command to view a snapshot of Linux system processes and their processor and memory consumption. Figure 3-19 shows results of the top command.

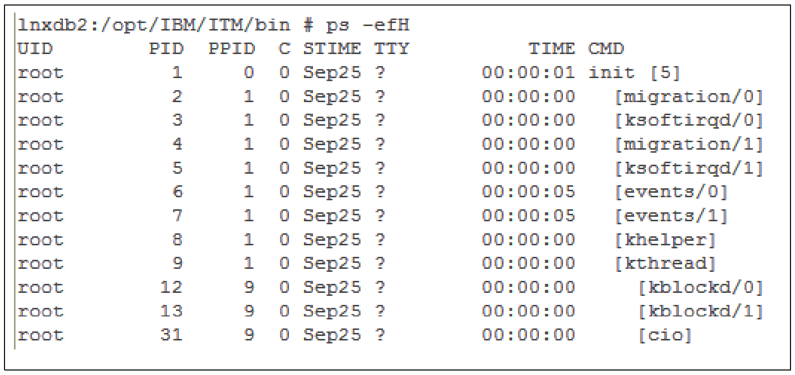


```
top - 14:47:33 up 20:41,  2 users,  load average: 0.05, 0.03, 0.00
Tasks:  92 total,   2 running,  90 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.2%us,  0.2%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    505168k total,   457708k used,    47460k free,    77804k buffers
Swap:   719896k total,       68k used,   719828k free,   160520k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
20090 root      16   0  2640 1300  980 R    0  0.3   0:00.07 top
    1 root      16   0   848  316  264 S    0  0.1   0:01.36 init
    2 root      RT   0     0    0    0 S    0  0.0   0:00.04 migration/0
    3 root      34  19     0    0    0 S    0  0.0   0:00.01 ksoftirqd/0
    4 root      RT   0     0    0    0 S    0  0.0   0:00.04 migration/1
```

Figure 3-19 Linux command - top

3.6 Process status (ps) command

The **ps** command displays running process information. The display is a one-time display, with no automatic updating, that shows UID, process ID, start date and time, and process name. Use the **man ps** command to see available options for parameters and display format. Figure 3-20 shows the **ps** command with the option efH for listing every process, full format, hierarchy structure.



```
lnxdb2:/opt/IBM/ITM/bin # ps -efH
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Sep25 ?        00:00:01 init [5]
root         2     1  0 Sep25 ?        00:00:00   [migration/0]
root         3     1  0 Sep25 ?        00:00:00   [ksoftirqd/0]
root         4     1  0 Sep25 ?        00:00:00   [migration/1]
root         5     1  0 Sep25 ?        00:00:00   [ksoftirqd/1]
root         6     1  0 Sep25 ?        00:00:05   [events/0]
root         7     1  0 Sep25 ?        00:00:05   [events/1]
root         8     1  0 Sep25 ?        00:00:00   [khelper]
root         9     1  0 Sep25 ?        00:00:00   [kthread]
root        12     9  0 Sep25 ?        00:00:00     [kblockd/0]
root        13     9  0 Sep25 ?        00:00:00     [kblockd/1]
root        31     9  0 Sep25 ?        00:00:00     [cio]
```

Figure 3-20 Linux command - ps

3.6.1 System status (sysstat) tool

This is a widely used package of Linux tools that collect system data. You can downloaded the package for free at the following URL if it is not included in your Linux system:

http://sebastien.godard.pagesperso-orange.fr/

You need to compile the sysstat package after installing it on Linux. The package contains the following components, but this is not a complete list:

- **sadc** data monitor: Stores data in a binary file (suitable for permanent system monitoring and detailed analysis)
- **sar** reporting tool: Reads binary file created with sadc and converts it to readable output
- **iostat**: Processor and I/O statistics for partitions and devices
- **mpstat**: Processor utilization per processor or combined
- **pidstat**: Statistics for Linux tasks including CPU, I/O, and memory

The reporting tool, sar, prints this information:

- Process creation
- Context switching
- Memory usage
- Swapping
- All/single CPU utilization
- Network utilization and errors on device level
- Disk I/O overview on device level.

Note: We recommend that you use sadc to periodically collect performance data when the system is running well and no problems are noted. You can then compare this healthy data with that collected during troubleshooting. Having these comparables will help you identify performance problems.

sadc has many parameters (see man pages). We recommend that you use the **-d** parameter, which includes disk device information and increases the outfile size. The outfile contains information in binary format.

sar reads the outfile and creates reports as requested. You can specify a start time and end time. Example 3-1 shows **sar** with parameter A specified (that is, all information).

Example 3-1 sadc displaying collected data from binary outfile using sar

```
/usr/lib64/sa/sadc -d [interval in seconds] [outfile]

sar -A -f [outfile]
```

Note: You can do data collection over a longer period of time, for example, an entire day, to capture peaks. If so, keep in mind the amount of data that you will need to process or transfer later. The outfile is smaller if you omit disk information or collect the data less frequently.

For a more detailed description, see this website:

http://public.dhe.ibm.com/software/dw/linux390/perf/Linux_standard_monitoring_tools.pdf

3.6.2 Netstat

This command gathers information about the Linux networking subsystem, such as network connections, interface statistics, and routing tables (Example 3-2). The first parameter determines which information is printed. To view details, use the **netstat** man page.

Example 3-2 Typical usage of netstat command

```
Display statistics for each protocol:
#netstat -s

Display statistics for all network interfaces:
#netstat -i
```

3.6.3 OProfile tool

OProfile is an open source profiler that offers profiling of all running code on a Linux system, including the kernel, shared libraries, and application binaries, and so on. It also provides a variety of statistics at a low overhead, varying from 1 - 8%, depending on the workload. It is released under the GNU GPL. OProfiler consists of a kernel driver, a daemon for collecting sample data, and tools for to generate reports.

OProfile uses CPU performance counters to count events for all of the running code, and aggregates the information into profiles for each binary image. Currently, System z does not support this type of hardware performance counters, but you can use the timer interrupt instead. Example 3-3 illustrates typical OProfile usage on a Linux guest running on z/VM.

Example 3-3 OProfile on Linux on System z

```
gunzip /boot/vmlinux-2.6.32.12-0.7-default.gz           1

opcontrol --vmlinux=/boot/vmlinux-2.6.32.12-0.7-default 2
opcontrol --start                                       3

<NOW OPROFILE COLLECTS DATA, DO THE TEST>               4

opcontrol --shutdown                                    5
opreport                                                6
```

Table 3-3 describes the bolded numbers on the right side of the OProfile screen (Example 3-3).

Table 3-3 Descriptions of OProfile displayed numbers

Number	Description
1	Unzip the vmlinux file if none exists.
2	Specify the vmlinux file for the running kernel.
3	Start the OProfile daemon and begin profiling.
4	Run the testcase while data gets collected.
5	Stop the OProfile daemon and stop profiling.
6	Product symbol or binary image summaries.

The OProfile report allows you to analyze the performance of a target module and adjust tuning accordingly. Figure 3-21 shows the first three functions of the application mcf_bas.z_Linux consume most of the CPU time. You can improve application performance by focusing on such intensely used functions.

```
CPU: CPU with timer interrupt, speed 0 MHz (estimated)?
Profiling through timer interrupt
vma         samples  %        app name           symbol name
80002840    5862     34.8970  mcf_base.z_Linux   price_out_impl
800012c8    5221     31.0811  mcf_base.z_Linux   refresh_potential
80003cb4    4398     26.1817  mcf_base.z_Linux   primal_bea_mpp
80003b60    408      2.4289   mcf_base.z_Linux   sort_basket
0001a67c    345      2.0538   vmlinux            default_idle
800013d8    138      0.8215   mcf_base.z_Linux   flow_cost
800033bc    98       0.5834   mcf_base.z_Linux   update_tree
800020f8    88       0.5239   mcf_base.z_Linux   dual_feasible
800036a4    72       0.4286   mcf_base.z_Linux   primal_iminus
8000323c    40       0.2381   mcf_base.z_Linux   write_circulations
80002720    24       0.1429   mcf_base.z_Linux   insert_new_arc
```

Figure 3-21 Example of OProfile output

This list presents a description of the oProfile output:

- *Virtual Memory Area* (vma) is a contiguous area of virtual address space. These areas are created during the life of the process when the program attempts to memory map a file, links to a shared memory segment, or allocates heap space:
- *Samples* is the number of samples for the symbol.
- *%* is the percentage of samples for this symbol relative to the overall samples for the executable.
- *app name* is the application name to which the symbol belongs.
- *symbol name* is the name of the symbol that was executed.

For more details about OProfile commands, see this website:

http://oprofile.sourceforge.net/docs/

OProfile hardware sampling support is not an available option at the time of writing, but it is described in documentation related on kernel 2.6.39 as “Oprofile hardware sampling support.” You can use OProfile hardware sampling if your Linux system runs in LPAR and if the machine provides a hardware sampling support, which System z10 and later provide. For more information, see the documentation on DeveloperWorks at this website:

http://www.ibm.com/developerworks/linux/linux390/documentation_dev.html

3.7 Other monitoring tools

You can also examine other performance monitoring software products that provide similar features and functions as the ones that we have discussed. This is not a complete list, but these are some of these products:

- Velocity zVPS
- BMC Mainview for Linux
- Computer Associates Explore Performance Management for z/VM

3.7.1 Monitoring with zVPS

zVPS is a z/VM and Linux performance suite of products by Velocity Software also remarketed by Computer Associates. This product suite includes the tools zMAP, zMON, zTCP, and zVWS.

zVPS collects and analyzes data from your Linux servers, network, and z/VM systems in a consistent and integrated format. Data is continually collected, presented in real-time, and stored in a single performance database. Automatic analysis triggers an alert when exceptions occur. You can generate reports for long-term trend analysis and capacity planning and access all information using your CMS terminal or web browser.

zVPS uses SNMP as its primary data source for network data. This standard data source provides load information on the transport layer, the internet layer, and every interface on each node. zVPS monitors all SNMP-compliant network nodes, including OS/390® (MVS™), z/VM, Linux, OS/2, Windows, Sun/OS, DEC/OS, Macintosh, most Unix-based networks, routers, and printers.

You can access zVPS real-time displays, historical reporting, and trending reports on your intranet. Your data is secured by an implementation of Secure Sockets Layer (SSL) that is included with the product.

3.7.2 BMC Mainview

BMC Mainview for Linux is another performance management tool available for the z/VM Linux environment.

3.7.3 Explore by Computer Associates

Computer Associates Explore Performance Management for z/VM is part of a suite of products from Computer Associates for managing Linux systems on z/VM.

3.8 Capacity planning software

Successful capacity planning involves projecting future needs based on past trends and business plans for growth. Although hardware costs continue to decline, software and personnel costs are rising. This makes capacity planning critical because software and personnel costs are linked to hardware capacity.

Capacity planners judge when to buy more hardware based on current throughput and performance, plus growth expectations for the IT workload. Performance is thus a key indicator when predicting the timing for a processor upgrade. High processor utilization is desirable from a cost viewpoint, but only as long as performance does not suffer.

Capacity planners need sophisticated tools and good modeling software that will help them design hardware upgrades before poor system performance affects business growth. IBM offers quality modeling tools for System z environments such as z/VM Capacity Planner and the Tivoli Performance Modeler. You can use the IBM z/VM Planner for Linux Guests on IBM System z Processors if your System z processor has only z/VM and Linux images. IBM employees and authorized IBM Business Partners can use the tool with their customers.

If your System z process is running a combination of z/VM, Linux, and z/OS images, you can use a combination of the IBM z/VM Planner for Linux Guests on IBM System z Processors and Tivoli Performance Modeler.

3.8.1 IBM z/VM Planner for Linux Guests on IBM System z Processors

The z/VM Planner for Linux Guests on IBM System z Processors (z/VM-Planner) is a PC-based productivity tool that runs on MicroSoft™ Windows™. z/VM-Planner is designed to provide capacity planning insight for IBM mainframe processors running various Linux workload environments as guests under VM. Input consists primarily of z/VM guest definitions and capacity requirements for each intended Linux guest. The total guest capacity requirement is calculated based on the guest definitions and, because it is probable that all guests do not have peak utilization at the same time, a user-specified complementary peak percentage is used to calculate the true combined capacity requirements of the z/VM guests being sized. The tool also models z/VM processor requirements when merging new guests into existing VM images. The resulting guest capacity requirement is combined with that of VM to support the entire complement of guests.

The z/VM-Planner produces several graphs that depict the modeled system. The graphs illustrate the size of processor needed to support the modeled workload, the percent of that processor used by each application in the workload, and the estimated minimum and maximum processing capacity needed for the workload. Figure 3-22 shows the minimum and maximum capacity needs for a modeled workload with five Linux guests running a mixture of web applications, databases, and file servers. Contact your IBM representative to learn more about this tool.

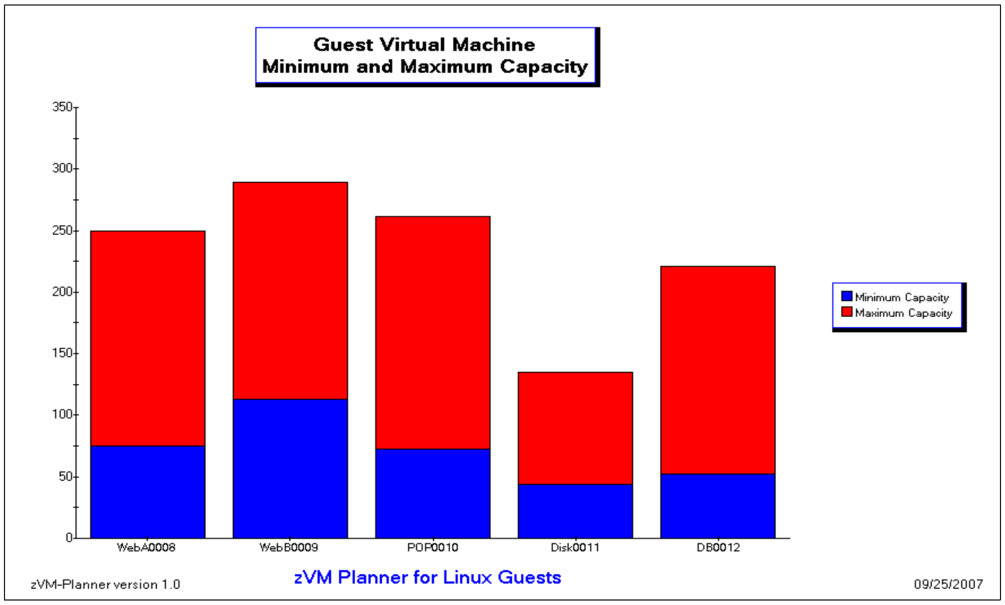


Figure 3-22 z/VM-Planner modeled system graph

3.8.2 IBM Tivoli Performance Modeler

IBM Tivoli® Performance Modeler for z/OS® V2.3 is a PC-based performance modeling and capacity planning tool that runs on MicroSoft™ Windows™. IBM Tivoli Performance Modeler for z/OS V2.3 is designed for system management on the IBM eServer® zSeries® and S/390®. Basic CPU utilization is no longer sufficient information for performing capacity planning due to growing operating system complexity and the huge impact of responding to workload changes. IBM Tivoli Performance Modeler for z/OS V2.3 can model the impact of changing and can be as portable as your PC. The tool models the following:

- Number and speed of CPUs
- Disk I/O response times
- Paging rates, auxiliary, and expanded memory paging
- LPAR definitions and parameter changes

The Tivoli Performance Modeler collects input from z/OS images, including system and application performance data, as a basis for the modeling program. You can then make changes to the type of processor and modify the workload configuration to create a new model. The Modeler then creates a series of graphs to depict the modeled workload. Figure 3-23 shows a complex workload of batch, database, and online applications.

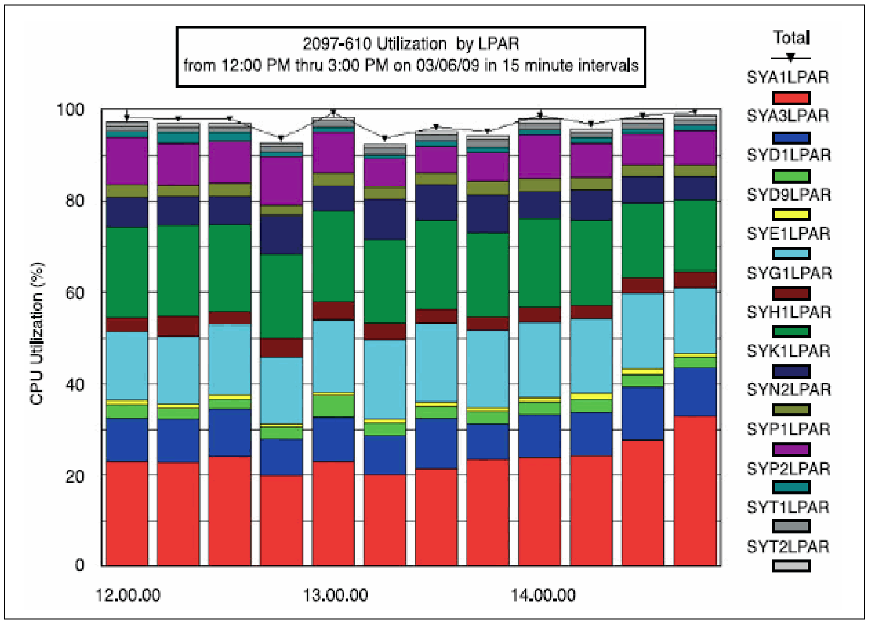


Figure 3-23 Tivoli Performance Modeler graph of modeled systems

4

z/VM storage concepts

In this chapter we discuss z/VM storage-related concepts such as z/VM storage hierarchy, z/VM storage allocation, virtual storage for Linux guests, z/VM storage management, and paging and spooling.

4.1 z/VM storage hierarchy

Both z/VM and Linux use virtual storage or virtual memory. However, z/VM generally uses the term *storage*, whereas Linux generally uses *memory*. We use the term storage in this chapter to explain z/VM-related concepts, but in other chapters referring to Linux, we use the term memory. Both terms are identical.

Virtual storage allows programs and data to share real, or physical, storage at the same time. A program accesses a small amount of storage even over a period of time. It is unlikely that a program will access more than a fraction of the total storage that has been assigned to it. A virtual storage mechanism called paging ensures that storage actively being used by a program is in real storage. Paging also temporarily saves storage not being actively used to expanded storage or disks, thus freeing real storage for active programs. Both z/VM and Linux manage storage in 4 K pages.

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

- Main storage

  Directly addressable and fast-accessible by user programs. Both data and programs must be loaded into main storage, from input devices, before they can be used by the processor. The maximum size of main storage is restricted by the amount of physical storage.

- Expanded storage

  This also exists in physical storage, but is addressable only as entire pages. Physical storage allocated as expanded storage reduces the amount for main storage. Expanded storage is optional, and its size is configurable.

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

- Paging space

  This resides on DASD. z/VM uses paging space when paging demands exceed expanded storage capacity.

Figure 4-1 shows the relationship between the types of z/VM storage.

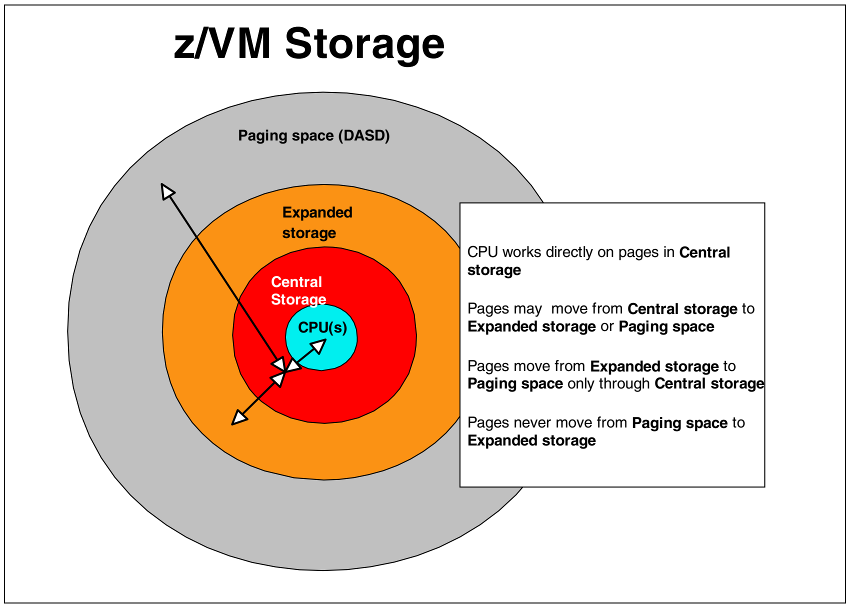

z/VM Storage
Paging space (DASD)
Expanded storage
Central Storage
CPU(s)
CPU works directly on pages in **Central storage**
Pages may move from **Central storage** to **Expanded storage** or **Paging space**
Pages move from **Expanded storage** to **Paging space** only through **Central storage**
Pages never move from **Paging space** to **Expanded storage**

Figure 4-1 Relationship between z/VM storage types

4.2 Relationship of z/VM storage types

The combination of main storage, expanded storage, and paging spaces forms the z/VM virtual storage address space. As illustrated in Figure 4-1, pages move within z/VM virtual storage to accommodate demands as follows:

- z/VM guests run in main storage.

  Guests execute in main storage when dispatched. Not all guest pages need to reside in main storage when running. Inactive pages can reside in expanded storage, in paging spaces, or in both.

- Paging occurs between main storage and expanded storage.

  z/VM moves inactive guest pages to expanded storage as demand for main storage increases. When those pages become active, z/VM moves them back to main storage.

- Paging also occurs between main storage and paging space.

  z/VM pages between main storage and paging space if no expanded storage is configured. z/VM also pages between main storage and paging space if the paging demand exceeds the expanded storage capacity.

- 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 storage.

4.3 Allocating z/VM storage

Overcommitted storage is a normal and desirable situation on z/VM, as we discussed in 1.2.1, “Overcommitting resources” on page 5. Storage overcommitment allows z/VM to provide more total resource 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? However, we continue to recommend that you to configure z/VM with expanded storage because expanded storage often results in a more consistent and better response time.

Expanded storage improves response time in these ways:

- Paging will probably happen in a z/VM system.

  The logic for allocating all physical storage as main storage is that paging only occurs if there is a shortage of main storage, so expanded storage only increases this possibility. However, overcommitted storage on z/VM is a normal and healthy practice. Consequently, we recommend that you prepare for paging rather than try to prevent it.

- z/VM paging algorithms are tuned for expanded storage.

  VM paging algorithms evolved around a hierarchy of paging devices where expanded storage is a high-speed paging device, whereas DASD is slower. It is more efficient to move pages from main storage to expanded storage, rather than move pages from main storage to paging DASDs.

- Two GB limitation.

  VM only supported 31-bit architecture before z/VM 3.1.0, which meant that all of the programs and data had to reside in the 2 GB address space. Sixty-four-bit support began with release 3.1.0, however parts of CP must still reside below the 2 GB line until the release of 5.3. This created contention in the past but is no longer an issue in newer z/VM releases.

  Note: You can view storage contention by using the QUERY FRAMES command with z/VM Performance Toolkit or OMEGAMON.

  Generally, you can configure expanded storage to 10% of the physical storage allocated to z/VM. Many systems do not need more than 2 GB of expanded storage regardless of the total storage available. You might need to allocate a higher percentage of processor storage as expanded storage if z/VM is paging more to DASD than to expanded storage.

  As we have noted, expanded storage might result in more paging, but it often results in more consistent and better response time. Expanded storage can also act as a buffer for more active users as they switch between working sets.

Overcommitting storage

You can realize performance gains by overcommitting real storage when expanded storage is available. Sometimes you can benefit by allowing the scheduler to treat a fraction of expanded storage as real storage. Do this by using **`set srm storbuf`** to overcommit real storage. Now the real storage available to virtual machines trying to enter the dispatch list appears larger even though the virtual machine working sets are still computed as before. This way more virtual machines are allowed into the dispatch list. This can result in higher paging rates, but the benefit of reducing the eligible list and moving users into the dispatch list can offset the increase. You can manage this by monitoring the paging rate against the response time and command rate, and then adjusting the STORBUF setting accordingly.

How to overcommit storage

Follow these steps to overcommit your storage space:

1. Specify one or more of the SET SRM STORBUF parameters to over 100%. We suggest parameter settings of 125 105 95. This command sets the parameters below or the scheduler.
2. The sum of the working sets for all Q0, Q1, Q2, and Q3 virtual machines must be less than or equal to 125% of real storage.
3. The sum of the working sets for all Q2 and Q3 virtual machines must be less than or equal to 105% of real storage.
4. The sum of the working sets for all Q3 virtual machines must be less than or equal to 95% of real storage.

The latest z/VM release contained revised STORBUF values that increased the overcommitment of real storage.

For more tips about storage configuration, see this website:

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

4.4 How z/VM uses storage

Sixty-four-bit support, first provided in z/VM 3.1.0, has improved through z/VM 5.4 and z/VM 6.1. We discuss the storage use below the 2 GB-line throughout these releases.

z/VM supports 64-bit and programs and data saved above 2 GB of main, or real, storage, but some areas of the control program (CP) are limited to 31-bit (2 GB) addressing. Throughout VM Versions 3.1.0 to 5.1.0, all CP code and data structures have to reside below the 2 GB line, and most CP code continues to use 31-bit addressing. A guest page can reside above 2 GB, execute, and reference data correctly. But if a page is referenced by CP or requires CP processing, it must be moved to a frame below the 2 GB line to access the 31-bit CP code. I/O channel programs and data, both SSCH and QDIO, simulation of instructions, and locked pages (for example, QDIO structures for real devices) are examples of such pages.

For example, if a guest is executing an I/O while the channel programs and data reside in a guest page above 2 GB, CP needs to perform page translation to process the virtual I/O. During the translation, the guest page is moved below 2 GB in main storage and locked until the I/O completes. The page remains in the main storage until it is stolen or released. This can cause storage contention below the 2 GB line. If that happens, you need to move the pages out of main storage and bring them back again quickly.

Note: A page can be stolen during the CP stealing process that starts when the number of available pages below 2 GB is too low. During the CP stealing process, pages below 2 GB are moved to expanded storage or paging DASDs. This is another reason to configure expanded storage in your system, because it is faster than paging DASDs.

A page in main storage can be released explicitly after the guest logs off or after a system reset.

CP runs its own address space, system execution space (SXS), that can be up to 2G. Since z/VM 5.2, the SXS is no longer identity mapped, which allows pages in the SXS to reside in any part of storage, including the frames above the 2 GB line. When CP references a guest page, it can map the page to a logical page in SXS without moving it below 2 GB. Also in z/VM 5.2, the CP control block that maps real storage was moved above 2 GB, and some CP modules were changed to 64-bit addressing. All of these changes reduce the conditions that require moving pages below the 2 GB line.

Page Management Blocks (PGMBKs) in earlier z/VM releases had to reside below 2 GB in real storage. PGMBK includes the page tables and related page management tables for a 1-megabyte segment of virtual storage. Each PGMBK is 8 K in size and pageable. When it resides in storage, a PGMBK is backed by two contiguous frames below 2 GB in main storage. If at least one page of the 1 megabyte segment of virtual storage is backed by one of the frames in real storage, the PGMBK must reside in main storage. Use OMEGAMON or z/VM Performance Toolkit to check PGMBK. For more details, refer to *z/VM Performance Toolkit Guide*, SC24-6156.

z/VM 6.1 eliminated the PGMBK limitation. PGMBKs can reside above the 2 GB in real storage. This greatly increases the maximum amount of in-use virtual storage supported by VM. z/VM 5.2 supported 128 GB, but that was increased to 256 GB in release 6.1. A system with 256 GB of real storage can support 21.3 TB of in-use virtual storage, which is beyond the CP storage management design limit of 8 TB. Additional improvements since release 5.4 are better management of contiguous frames, and more efficient search for single and contiguous frames on the available list.

When z/VM needs space to bring in guest pages and there are no open slots available, the system scans pages in storage for candidates to page out. z/VM scans in three phases as part of its normal storage management. Scan 1 moves pages that are obviously not needed. Scan 2 searches deeper to find more unneeded pages, and scan 3, otherwise known as emergency scanning, performs the most intensive moving of pages, stealing them if necessary. Emergency scanning in earlier releases of z/VM indicated that the system was critically short of storage. In current releases, emergency scanning does not indicate a critical storage shortage and is not a cause for concern.

For more z/VM performance details, see this website:

http://www.vm.ibm.com/perf/reports/zvm/html/

4.5 How Linux guests see virtual storage

Virtual storage consists of main storage, expanded storage, and paging space. Linux guests in z/VM see storage 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 storage and can be moved out of main storage to accommodate demand. z/VM can move a Linux guest's pages without notifying the guest.

4.5.1 Double paging

z/VM moving guest pages out of main storage can lead to a double paging effect (Figure 4-2).

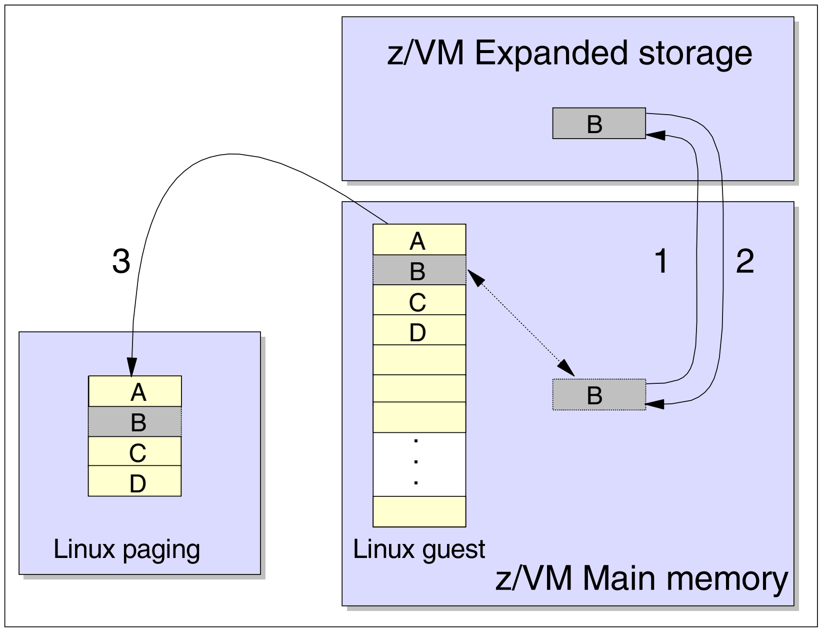


Figure 4-2 Illustrating the double paging effect

Figure 4-2 depicts storage pages used by a Linux guest. The numbered arrows show the event sequence leading to double paging:

1. After a period of inactivity, page B, mapped to a running Linux guest, is moved from main storage to expanded storage. The page is now available for demand page-in. z/VM has stolen the page without informing the Linux guest.

   Note: If no expanded storage was configured, z/VM will move the page to paging space.

2. Page B is moved back into main storage in response to a page-out by the Linux guest. To alleviate a storage constraint, Linux identifies inactive pages, here page B, and moves them to the swap device. The Linux guest page faults when requesting page B. This, in turn, causes z/VM to page-in storage 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 moved back into main storage, Linux pages it out to its swap device. In reality, z/VM has moved the page back into main storage simply to allow Linux to move it out again.

Managing double paging

Double page faults are not unique to Linux and can also occur in z/OS running under z/VM. The effect is a result of two parties separately managing memory at the same time. One solution is to not allow one party to page, but there are other options, such as these:

- Set the Linux guest virtual machine size small enough for z/VM to keep in main storage.

  Use working set size estimation techniques to determine how much storage to assign a Linux guest and then allocate slightly more storage than the working set size. Double paging might still occur if many smaller Linux guests compete for z/VM main storage.
- Set the virtual machine size large enough so that Linux does not try to swap.

  This option can lead to frequent page faults. To decrease page faults, use PAGEX/PFAULT, an asynchronous mechanism that allows z/VM to inform Linux that a requested storage page does not reside in main storage. Linux will then try to dispatch another process.
- Cooperative Memory Management (CMM).

  When you use CMM, the Linux guest passes storage information to z/VM. z/VM then steals pages based on the current usage information. If the Linux guest accesses pages removed by z/VM, discard faults are delivered and the Linux guest recreates the content of the discarded page. For more details about CMM see, 6.9, “Cooperative Memory Management (CMM)” on page 99.
- Collaborative Memory Management Assist (CMMA).

  CMMA is the latest solution to solve the double paging issue. The pages in use are flagged when allocated, making it easier to determine whether they can be reused. Unlike the CMM solution above, the Linux guest completely controls CMMA. For details about CMMA, see in 6.10, “Collaborative memory management assist (CMMA)” on page 100.

4.5.2 Allocating storage to z/VM guests

You can assign z/VM virtual machines storage in the user directory entry. Two values in the directory entry specify storage for each use (Example 4-1):

- The first value is the initial storage size of the guest virtual machine when it logs on.
- The second value is the maximum storage size that can be set after the guest has logged on.

Example 4-1 A user directory entry for Linux LXNMR1

```
USER LNXMR1 LNX4ME 512M 2G G
   INCLUDE IBMDFLT
   IPL CMS
   MACHINE XA
   OPTION LNKNOPAS APPLMON
   NICDEF C200 TYPE QDIO LAN SYSTEM VSWITCH1
   MDISK 0191 3390 141 20 LX6U1R MR
   MDISK 0101 3390 1001 1000 LXCD3B
   MDISK 0201 3390 0001 3338 LXCD3A
```

Storage is reset and the initial program load (IPL) reoccurs when the guest changes its virtual machine size. Changing a virtual machine size is usually done by a human user, not by an operating system running in a virtual machine. Table 4-1 lists additional ways to change storage allocation.

Table 4-1 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.

4.5.3 VDISKs

VDISKs are virtual disks, emulated disks that z/VM creates in virtual storage. VDISK is backed by a storage address space instead of a DASD. VDISKs are fast because they reside in main storage. Typically, Linux guests use VDISK for a fast swap device. See 7.4.6, “The advantages of a VDISK swap device” on page 113, for details.

You can reduce latency in the Linux swap I/O by using VDISK as a swap device provided that your system has sufficient storage. VDISK might be appropriate for your system if it is storage-constrained below 2 GB but rich above that.

4.5.4 Minidisk cache

Minidisks cache (MDC) is a data-in-storage technique that improves performance by decreasing the I/O to DASD required for minidisk I/O. MDC decreased DASD I/O at the expense of real or expanded storage. Real and expanded storage for MDC decreases when paging to DASD increases, which might increase paging I/O in z/VM. However, this type of paging in z/VM is normal and healthy, which makes using MDC worthwhile.

All minidisks that meet the requirements for minidisk cache are enabled by default. The CP arbiter function determines how much real or expanded storage can be used as minidisk cache and paging. If the arbiter is favoring paging or minidisk cache more than is optimal for the system, it can be biased for or against minidisk cache by using the SET MDCACHE STORAGE BIAS or SET MDCACHE XSTORE BIAS command

The z/VM minidisk cache is implemented with a system cache where the storage cache is shared with all the users on the system. CP enforces a fair share limit to prevent one user from negatively impacting another user on the same system. I/O completes without inserting the data into the cache when a user reaches the fair share limit. You can override the fair share limit by using the NOMDCFS operand on the OPTION statement in the user directory entry. We recommend that the key users, such as server machines or guests that do I/O on behalf of many users, turn off the fair share limit.

Make sure that you configure paging appropriately because using minidisk cache usually increases paging. We recommend that you do not mix paging space with other types of data, including spool space. We also suggest that you do not allow page block size to fall below 10, but instead balance the paging I/O over multiple control units.

4.6 Managing z/VM storage

You can influence how z/VM manages storage in several ways. Generally, we recommend that you use options that enable resource sharing to maintain overall system performance. Use the following options to influence storage:

- Update system software to the current level.

  z/VM storage management improves with each release, so using recent releases will improve system performance. Read the z/VM performance report for details about storage management improvements for each release at this website:

  `http://www.vm.ibm.com/perf/reports/zvm/html/index.html`

- Make sure that there are enough paging DASDs with open (that is, not busy) paths.

  This step makes paging as fast as possible. Use multiple devices over as many DASD control units as possible to permit overlapped I/O. Use entire dedicated volumes for z/VM paging instead of fractional volumes. That is, do not mix page spaces with other space, such as user, spool, tdisk, and so on. Make sure to define enough space so that you can conduct block paging. Try to provide at least twice as much DASD paging space as the sum of the Linux guests' virtual storage sizes.

- Use expanded storage for paging.

  As we have noted, expanded storage can improve overall performance. A good starting point is to define 10% of the real storage as expanded storage. If your system is not constained by 2 GB line, 2 GB of expanded storage is usually enough.

- Use shared segments.

  Typically, the 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 the creation of a Linux NSS later in this book. By using a NSS, only one copy of the Linux kernel is saved in main storage, and the pages making up this kernel can be shared with many virtual machines. This decreases storage requirements, because there is not a separate kernel for each guest.

- Explore discontiguous saved segments (DCSS) when appropriate.

  One of the unique advantages of z/VM is the ability to share segments among virtual machines. There are a number of segments that might be predefined on your system. Any segments not being used should not be loaded into main storage. Use the QUERY NSS MAP command to determine how many users are using a segment. There is only one set of PGMBKs in the main storage when a segment is shared by several virtual machines. This can reduce the storage requirement if the content in the segment is required by a large number of Linux guests.

- Adjust System Resource Management (SRM) controls.

  SRM directly controls how z/VM manages virtual storage. SRM parameters are discussed in 8.4.2, "Global System Resource Manager (SRM) controls" on page 142.

- Make sure to properly retrieve and store system operation information records.

  z/VM collects various accounting, error, and symptom records during operation. This data is stored in main storage 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.

  Available system storage can be greatly reduced if these virtual machines stop retrieving the data. For details about information collection, see *z/VM V5R3.0* z*/VM: System Operation*, SC24-6121.

- Use the CP SET RESERVED command to reserve pages for an important guest.

  The CP SET RESERVED command reserves real storage pages to a virtual machine. This command does improve performance for a specified guest, but use it carefully, as it can diminish system performance.

4.7 Paging and spooling

z/VM uses specially allocated DASD space for storage that is available to all virtual machines on the system and that has distinct uses. CP owns and controls this DASD space.

We have noted that paging DASD space is used for virtual machine paging, along with main storage and expanded storage. Because the data in paging space is temporary, all the paging data is discarded when z/VM is shut down.

Follow these guidelines to set up the z/VM system for paging:

- Provide at least twice as much DASD paging space as the sum of the Linux guests' virtual storage size, minus the main memory size. Use the CP QUERY ALLOC PAGE command to determine how much paging space is available.
- Use entire volumes for z/VM paging space. In other words, never mix paging I/O and non-paging I/O, such as user space, spooling, tdisk, and so on, on the same pack.
- Distribute the paging volumes over as many DASD control units as possible and implement a one-to-one relationship between paging CHPIDs and paging volumes. This makes the I/O concurrently occur to several paging DASDs through multiple paths.
- Ensure that the spool space disks space is only used 50% for better performance.
- Turn on NVS or DASDFW if the paging control units support them. Use the CP commands SET NVS ON or SET DASDFW ON.

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. Data sent to a virtual machine resides in the reader until deleted or read into a user's minidisk, much like a mail inbox. Data 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, for example, NSS files. This creates a common location for system code, rather than each virtual machine requiring individual storage space. Because the content of spool space is valid data, it is preserved across z/VM shutdowns and IPLs.

z/VM allows for temporary minidisk space along with DASD space that is owned by specific virtual machines, that is, minidisks. The virtual machine can allocate this space as needed from the disk space created on the system. This allows you to define disks of various sizes without considering their placement. However, this is intended only to hold data that is being processed or that does not need to be retained. All data in the disk space is discarded when z/VM is shut down. If you are using a temporary minidisk (tdisk), disable the minidisk cache for that minidisk.

5

Linux virtual memory concepts

In this chapter we discuss Linux memory management concepts and consider how these concepts affect Linux guests running in z/VM. We will examine memory components and management, aggressive caching, sizing, and hotplug memory.

5.1 Linux memory components

Memory, swap space, swapping and paging, page allocation, and aggressive caching are all basic components of Linus memory. We discuss these components in the following sections.

5.2 The Linux kernel

After booting, Linux's first task is to allocate memory resources and map all memory to the kernel address space, including kernel memory, user memory, and page cache memory, as described here:

- Kernel memory: This is where the actual kernel code is loaded and where memory is allocated for kernel-level operations, including these:
  - Scheduling
  - Process management
  - Signaling
  - Device I/O (including both to-disk and to-network devices)
  - Paging and swapping.

  On booting, pages that host the kernel code are marked as reserved and are not available for later allocation. The kernel is loaded at the bottom of the memory, and some space is reserved. The Buddy allocator controls the remaining memory.
- User memory is assigned by the Buddy allocator and used for the following:
  - Anonymous memory
  - Shared memory (System V or POSIX API)
  - Memory mapped files
- Page cache memory: The Linux kernel no longer contains separated page and buffer caches. Read, write, and map operations and direct access fill the page cache. The kernel manages pages with a page array structure and also hosts various data, for example:
  - Read and write operations of regular files
  - Memory mapped files
  - Anonymous memory

Note: Linux uses an aggressive caching policy to reduce I/O when allocating main memory, the theory being that memory is better utilized as cache rather than left unused and available. Read more about this theory in 5.4.2, "Aggressive caching in Linux" on page 56.

5.3 Linux swap space

Swapping is the process of moving an entire address space to a swap device, while paging is the process of moving pages to swap space. Linux uses the paging function rather than swapping. However, the terms *swapping* and *swap device* are used in this book even though Linux uses a paging algorithm.

During the initial system install, Linux creates a swap device and uses it during peak demand periods to temporarily store anonymous pages that are not frequently used. Those pages are then moved back into real memory whenever they are requested. See Chapter 7, "Linux swapping" on page 103, for details about swapping.

This is confusing, especially when you view a z/VM Performance Toolkit report that displays both paging and swapping rates (Example 5-1). To help you better understand the terms, we explain them in Table 5-1.

Example 5-1 Linux memory and activity details panel

```
Linux Memory Util. & Activity Details for System LINUX001

   Total memory size        493MB       Swap space size          140MB
   Total memory used        298MB       % Swap space used       0.06%
   Used for buffer          102MB       Swap-in rate               0/s
   Used for shared            0MB       Swap-out rate              0/s
   Used for cache           225MB       Page-in rate               0/s
   Total free memory         40MB       Page-out rate         25.866/s
```

Table 5-1 Swap and page values and descriptions

Value	Description
Page in	The number of disk blocks transferred to memory from disk. Also disk I/O in general.
Page out	The number of disk blocks transferred from memory to disk. Also disk I/O in general.
Swap-in rate	The number of memory pages paged in from swapspace.
Swap-out rate	The number of memory pages paged out to swapspace.

The activity reported for page-in and page-out rate could be any disk I/O, such as the flushing of cached file system pages, but it is not related to writing to the swap space. See more details on disk I/O in Chapter 10, "Tuning disk performance" on page 169.

Use the Linux **procinfo** command to view information about memory usage, pages, and swapping (Example 5-2).

Example 5-2 Sample output of the procinfo command

```
lnxwas:~ # procinfo
Linux 2.6.32.12-0.7-default (geeko@buildhost) (gcc 4.3.4) #1 SMP 2010-05-20
11:14:20 +0200 1CPU [lnxwas.]

Memory:      Total        Used        Free      Shared     Buffers      Cached
Mem:       1528716     1151164      377552           0      160580      515696
Swap:       719896           0      719896

Bootup: Mon Sep 12 12:36:57 2011    Load average: 0.24 0.24 0.17 1/192 29574

user  :       1:50:31.91   2.4%  page in :     525660  disk 1:    31692r   87484w
nice  :       0:00:00.40   0.0%  page out:     996588  disk 2:       49r       0w
system:       0:56:44.52   1.2%  page act:      85224
IOwait:       0:00:24.46   0.0%  page dea:          0
hw irq:       0:00:54.26   0.0%  page flt: 1360538180
sw irq:       0:04:50.11   0.1%  swap in :          0
idle  :   3d  1:40:38.33  96.0%  swap out:          0
steal :       0:08:27.63   0.2%
uptime:   3d  4:42:31.67         context :   36535516
```

```
irq  0:   4496001                     irq  1:  20086422
lnxwas:~ #
```

5.4 Linux memory management

Linux memory is divided into pages, with each standard page 4,096 bytes in size for the s390 architecture. System z supports page sizes of 1 MB, called large pages, which are organized separately (see 5.4.4, “Large pages” on page 58). Standard pages are described by a unique structure and are aggregated into memory zones. Sixty-four-bit Linux on System z allocates in both the DMA zone and the normal zone, while 31-bit Linux on System z allocates only in the DMA zone. Although zones can be grouped into nodes, System z does not currently use this function.

5.4.1 Page allocation

A busy Linux system with cache strategy uses most of the available memory for cache and processes, although not all necessary process pages are loaded in the main memory.

Page fault handling is the method for bringing required pages back into memory. When the system tries to access an unavailable page, it triggers a page translation check. If the system tries to write to an unavailable page, it triggers a protection exception check. In both cases, the system provides the address where the fault occurred. The Linux kernel page fault handler checks whether there is VMA including the address. If so, the page is loaded into memory by page-in, swap-in, or copy-on-write, and the stack can grow automatically if required.

5.4.2 Aggressive caching in Linux

Linux manages its memory without regard to the fact that it is running in a virtual machine. In other words, the Linux kernel, like most other UNIX systems, uses the always full concept of memory management. The kernel loads as much information (for example, applications, kernel, and cache) as possible into its perceived memory resources and caches it there. The goal is to reduce the number of disk accesses, because even the fastest disk devices are slower than memory access. In kernel 2.6.32, the read-ahead mechanism is caching data into the page cache while reading.

To illustrate Linux aggressive caching, we compare the memory usage of two Linux guests running a virtual machine, one in 64 MB and the other in 128 MB (Figure 5-1). Figure 5-1 displays memory usage for the kernel, the application, the caches, and the buffers over run time with the 64 MB and 128 MB sizes. Both run the benchmark workload over an 8-minute interval.

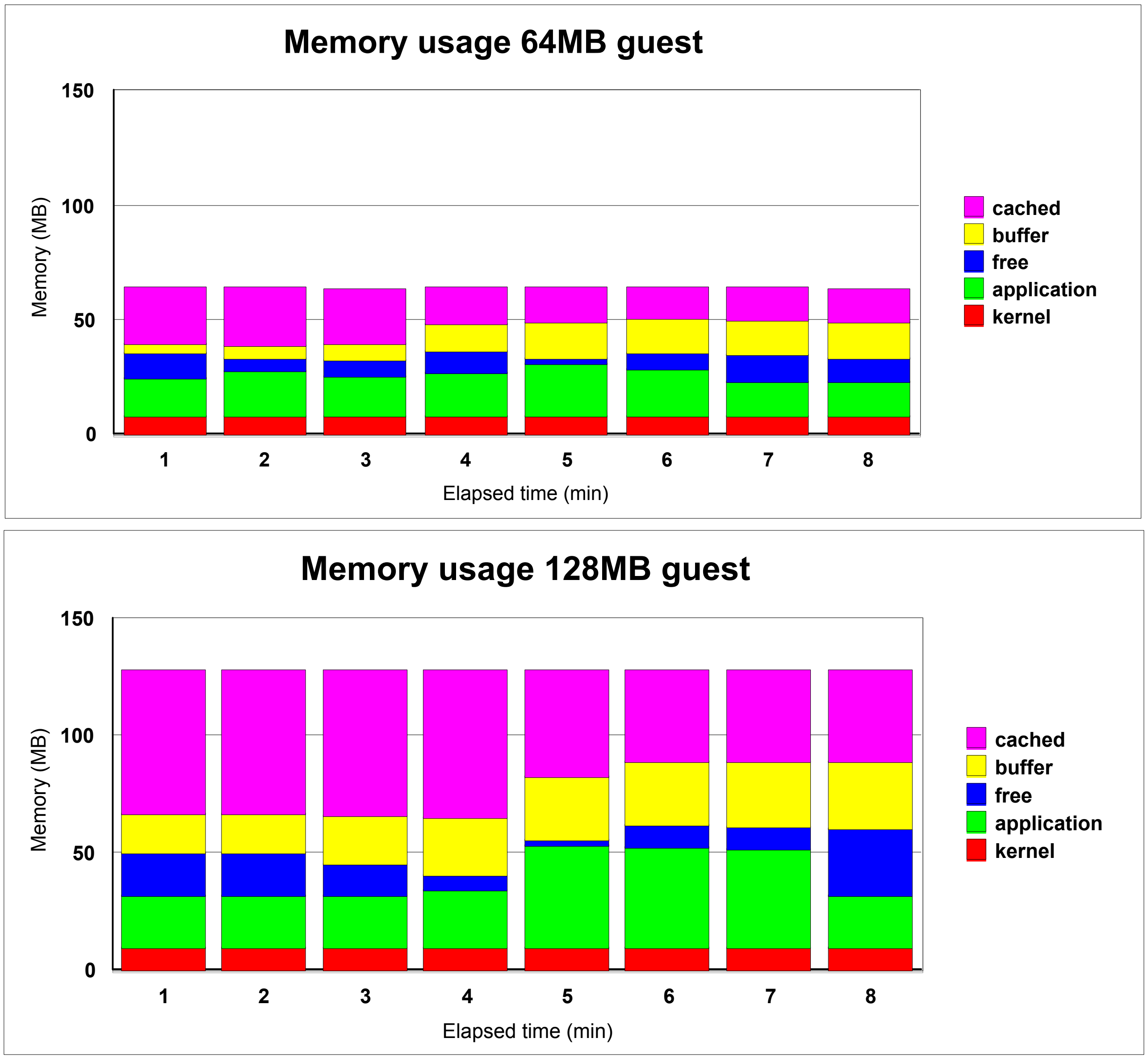


Figure 5-1 Linux memory usage in two Linux guests: 64 MB and 128 MB

The test run has nearly the same working set size, but it is not dependent on the available memory. The larger memory size is not allocated to free or available memory, but is used for buffers and caches.

In 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, there was no appreciable effect on server response time.

5.4.3 Page replacement

One goal of page replacement is to keep a number of pages free for further allocation. The page replacement strategy applies to anonymous pages in memory and in cache. Like other UNIX operating systems, Linux uses the two-handed clock mechanism. That is, a function scans portions of memory in an endless loop to check current page usage, as follows:

- The active list is checked for entries that can be added to the inactive list.
- The inactive list is checked for entries that can be reused after content is removed or written back.

Linux cleans memory pages in the following instances:

- The kswapd kernel thread runs.

  The kswapd sleeps most of the time. It gets invoked if the number of free pages is less than the pages_low variable. It runs until it reaches the pages_high mark or until there are no more pages to free.
- pdflush or per backing-device-based writeback (since kernel 2.6.32) is running.

  The purpose is to write back dirty or modified pages to disk. pdflush threads are started when the number of free pages is below a threshold. These threads write pages back to permanent devices so that the pages can be freed. After a specified time, the pdflush threads also wake up to write back pages that have been dirty for too long.
- Shrinking occurs in kernel slab caches.

  The pages in slab caches need separate algorithms.
- The allocation routine initiates direct freeing of pages.

  Allocation routing will free pages if the kswapd and the pdflush threads cannot keep pace with the memory requirements.

5.4.4 Large pages

Dynamic Address Translation (DAT) has long been used on mainframes to virtualize memory, with the advantage being that a process would view all memory as available. However, virtualization does consume some processor cycles in translating virtual addresses into real memory. Translation Lookaside Buffers (TLB) were introduced to reduce this translation overhead. After translation, TLB keeps the mappings of virtual to real addresses in slots so that they can be reused. This provides an advantage because programs often access the same memory frequently.

Memory is normally divided into chunks of 4 kB called pages. Even though server size has grown, the standard page size has remained at 4 kB. This has made it harder for TLB to track, and often results in large number of pages in memory.

Enhanced DAT provides a solution by supporting large pages of 1 MB. Enhanced DAT was added to the System z architecture with the release of z10. This specific backend for large pages was introduced with Novell SUSE Linux Enterprise server (SLES10 SP2) and Red Hat Enterprise Linux server (RHEL5.2.), where both used 2 MB pages requiring two consecutive pages in hardware. This changed with SLES11 and RHEL6, where the large page size was set to 1 MB in Linux.

IBM System z10 and Enterprise server provide large page hardware support available to Linux guests running in LPAR mode. Using large pages reduces the pressure on TLB, accelerates memory access, and can also reduce the page table memory used for large page mappings in Linux. On older hardware, or if the Linux guest is running as a z/VM guest, the large pages are emulated by software via contiguous normal pages of 4 KB size. There is no reduced TLB pressure in this software emulation mode, but the Linux page tables for large mappings are shared, which results in similar page table memory savings, as with hardware large pages. Example 5-3 shows the edat feature, provided that large page support is enabled in hardware.

Example 5-3 edat in /proc/cpuinfo

```
lnxwas:~ # cat /proc/cpuinfo
vendor_id       : IBM/S390
# processors    : 1
bogomips per cpu: 11061.00
features        : esan3 zarch stfle msa ldisp eimm dfp edat etf3eh highgprs
processor 0: version = FF,  identification = 12DE50,  machine = 2097
lnxwas:~
```

You can also use the **`mmap system call`** command, with the hugetlbfs file system, and SysV shared memory system calls for large page support. With Java, you need to specify the parameter -Xlp in order to use large page support in software or hardware. Applications on other architectures that support large pages need to be recompiled for s390 architecture.

Page table memory savings

We have noted that using large pages can reduce the page table memory required for large page mappings in Linux. Table 5-2 shows an example of possible savings in a comparison of 128 processes in a database application that are mapping 2 GB of shared large page memory.

Table 5-2 Possible table memory savings

Environment	Page table overhead	Memory consumption per process	Memory consumption overall
Without large page support	1 segment table, 2048 page tables	4112 kB	514 MB
With large page support emulated in software (pre z10, all z/VM guests)	1 segment table, 1 page table	18 kB	2.25 MB
With large page[a] hardware support (edat feature)	1 segment table	16 kB	2 MB

a. With hardware support there is an additional benefit by reduced TLB pressure. This comes with the large table segment entries.

For more details, see the kernel source here:

```
/usr/src/<linux-kernel>/Documentation/vm/hugetlbpage.txt
```

Note: The memory used by huge pages is locked and cannot be paged out.

5.5 Sizing Linux virtual memory

Optimal memory size is more important and has more impact on performance in a environment with virtualization layers and many shared resources than on a standalone server.

5.5.1 Influence of virtual memory size on performance

We ran a series of experiments in a three-tier environment (IHS server, WAS server, and DB2 database server) to show the influence of virtual memory size, and the heap with Java. We changed only the virtual Linux guest memory size and WAS server heap size. In this experiment we did not drive the Trade6 benchmark to its maximum results except to show that there is an optimal size for a workload. Too few or too many resources can harm the performance of a particular system, and even the overall performance of all the hosted systems could suffer. The transaction throughput was used to gauge how well Trade6 operates in the available Linux memory.

Note: In Java applications the heap size is important because eventually all memory is touched for removal of objects. The performance is severely degraded if the heap does not fit completely into main memory.

In these experiments, we ran the IHS server and the DB2 database server with the same configuration but did change the WAS server Linux's virtual memory and the WebSphere Application Server heap size. All the measurements were taken after a ramp up and warm up phase with 50 clients.

Table 5-3 shows environment configuration details. We configured all the servers with one CPU, IHS server, and DB2 database server, with a memory of 512 M and 1 G, separately. We changed the WAS server memory from 1G to 2G and 3G, with a different heap size of the WebSphere Application Server. Note that there is only 4G memory configured in the entire system, and the memory resource is consumed increasingly in these configurations.

Table 5-3 Details of the configurations

Linux	CPU	Memory	Java heap size
LNXIHS	1	512 M	N/A
LNXDB2	1	1024 M	N/A
LNXWAS	1	1024 M	512 M
	1	1024 M	768 M
	1	2048 M	1200 M
	1	3072 M	2200 M

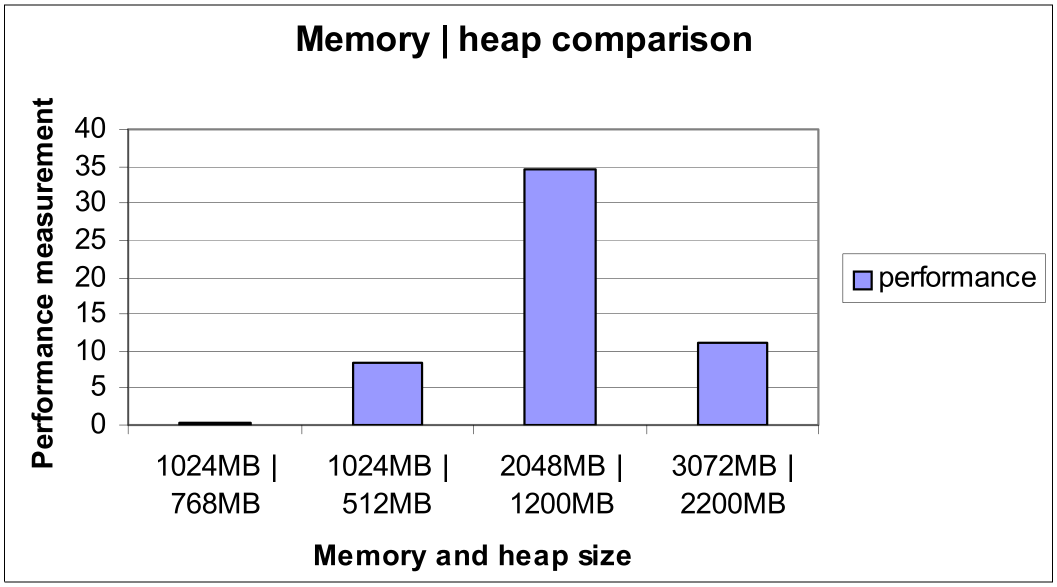


Figure 5-2 Performance of Trade 6 with varying virtual memory and heap size

We varied virtual memory and heap size of the application server, and Figure 5-2 shows the performance results. We drove only a moderate workload of 50 clients because these experiments illustrate the effect of virtual memory size on performance, and we used 1/response-time as the performance measurement.

In the first experiment, we ran 1024 MB of virtual memory, but the 768 MB heap size was too large to fit in main memory. Remember that the kernel and libraries also reside in the main memory. The result was that memory was eventually used completely and Linux started swapping. Java used the garbage collection routine, which can slow the system nearly to a halt.

In the second experiment, we ran 1024 MB of memory, but with a 512 MB heap size. In this case, swapping was eliminated and the performance was improved.

In the third experiment, we used 2048 MB of memory and 1200 MB heap size. Here performance was more improved than in the previous two experiments.

Finally, in the fourth experiment, we brought in z/VM in a memory overcommitment of 3072 MB of memory and 2200 MB heap size. Here z/VM starts to page moderately to expanded storage. The performance was less here than with 2048 MB memory but slightly better than with 1024 MB memory. However, overhead costs increased for this workload. We obtained good results even with more users because z/VM is used to working with an overcommitment ratio and handles paging well.

In conclusion, performance generally improves with additional resources. We saw significant differences when we increased both the virtual memory and the heap size from 1024 MB to 2048 MB in the third experiment. However, when virtual memory was increased to 3072 MB in the fourth experiment, performance dropped due to the memory management cost and overcommitment. This indicates that more resources do not always result in better performance. Make sure that you set the virtual memory appropriately to the working set size and to the entire environment. Otherwise, the performance can be negatively impacted, as we saw in the first experiment where the1024 MB application server was overloaded by the 768 MB heap size.

5.5.2 Considerations on sizing z/VM Linux guests

The Linux memory model has important implications for Linux guests running under z/VM, for example:

- z/VM memory is a shared resource.

  Although aggressive caching reduces disk I/O in favor of memory access, the cost of caching still must be considered. Cached pages in a Linux guest reduce the number of 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.

Choose the smallest memory footprint that has a minimal effect on performance when you size memory requirements for a Linux guest. If you do not, the remaining memory will be used for buffering and caching.

To keep shared memory alive, pin pages in memory. That is, do not make them available for replacing, swapping, or paging.

Use fast swap devices to reduce the penalty of occasional swapping that might occur in a smaller virtual machine. See Chapter 7, "Linux swapping" on page 103, for details.

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

Exceptions to the above rules are servers with special memory needs, such as these:

- Database servers

  Database servers maintain buffer pools to prevent excessive I/O to disk. Downsizing these bufferpools reduces performance.

- Servers running Java workload, for example, WebSphere Application Server

  The Java heap requires certain memory, and changing heap size can degrade performance even when no swapping occurs (as shown in the second experiment).

5.5.3 Managing hotplug memory

Hotplug memory is a method of dynamically adapting Linux memory size to suit requirements without rebooting the Linux system. Let us use an example of a database server that receives complex queries at the end of the month while running statistics. Running these complex queries quickly requires more database buffering, which requires more main memory. But during off-peak times, this server would still use memory allocations configured for the end of the month working set size. The result, in a virtualized environment, could be a high overcommitment rate, or surplus memory could be used for buffering and not made available to other systems.

Hotplug memory is divided into core memory and predefined memory. Core memory is the home of the Linux kernel and its data structures. It remains at the same size from boot to shutdown. Predefined memory is configured to the real hardware (LPAR) or virtual hardware (z/VM) to use as dynamic hotplug memory. Predefined memory must be set to reserved storage *o*n LPAR and to standby storage on z/VM.

Note: z/VM reserved storage cannot be used as hotplug memory in Linux.

LPAR supports hotplug memory since z/VM 5.4, but requires APAR VM64524. SLES11 SP1 and RHEL6.1 also support hotplug memory. Core memory is preserved at reboot and hotplug memory is freed.

To learn about setting up the environment and managing memory, see Device Drivers, Features, and Commands on SUSE Linux Enterprise Server 11 SP1:

http://public.dhe.ibm.com/software/dw/linux390/docu/l26edd01.pdf

5.6 Viewing Linux memory usage

For a quick view of Linux memory allocation, use the Linux **free -m** command, where **-m** reports memory size in megabytes. Example 5-4 illustrates memory use for a 128 MB virtual memory Linux guest.

Example 5-4 Observing Linux memory usage using free -m

```
lnxwas:~ # free -m
             total       used       free     shared    buffers     cached
Mem:           115        111          4          0          6         37
-/+ buffers/cache:         67         47
Swap:         7043          0       7043
lnxwas:~ #
```

Note these points about Example 5-4:

- The total memory of 115 MB is less than the total 128 MB virtual memory allocated to the Linux guest. The difference is the size allocated to the kernel.
- The total memory of 115 MB is equal to the used memory of 111 MB, plus the free memory of 4 MB.
- The used memory of 111 MB is equal to the buffer of 6 MB, plus the cached memory of 37 MB, plus the used buffers/cache memory of 67 MB.
- The used buffers/cache memory of 67 MB, plus the free buffers/cache memory of 47 MB, is equal to the total memory of 115 MB.
- The free buffers/cache memory of 47 MB is equal to the free memory of 4 MB, plus the buffer memory of 6 MB, plus the cache memory of 37 MB.

There are 47 MB of free memory available. The operating system expects to use the remainder as buffer/cache memory.

5.6.1 Kernel memory use at system boot time

To view kernel memory use at boot, use the **dmesg** command, which displays all the messages issued at boot time (Example 5-5). Note that 50720 KB is reserved for the kernel with remaining memory controlled by the Buddy allocator.

Example 5-5 dmesg command and memory related output

```
dmesg | less
...
On node 0 totalpages: 391424
```

```
  DMA zone: 5376 pages used for memmap
  DMA zone: 0 pages reserved
  DMA zone: 386048 pages, LIFO batch:31
PERCPU: Embedded 11 pages/cpu @0000000002581000 s12544 r8192 d24320 u65536
pcpu-alloc: s12544 r8192 d24320 u65536 alloc=16*4096
...
Dentry cache hash table entries: 262144 (order: 9, 2097152 bytes)
Inode-cache hash table entries: 131072 (order: 8, 1048576 bytes)
Memory: 1522144k/1572864k available (4382k kernel code, 0k reserved, 2114k data,
228k init)
...
HugeTLB registered 1 MB page size, pre-allocated 0 pages
...
```

5.6.2 Detailed memory use with /proc/meminfo

To view memory details, use **/proc/meminfo** to query the kernel driver (Example 5-6).

Example 5-6 Memory details from /proc/meminfo

```
lnxwas:~ # cat /proc/meminfo
MemTotal:        1528716 kB
MemFree:          364904 kB
Buffers:          171764 kB
Cached:           448904 kB
SwapCached:            0 kB
Active:           677560 kB
Inactive:         386960 kB
Active(anon):     443932 kB
Inactive(anon):        0 kB
Active(file):     233628 kB
Inactive(file):   386960 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:        719896 kB
SwapFree:         719896 kB
Dirty:                80 kB
Writeback:             0 kB
AnonPages:        443856 kB
Mapped:            94904 kB
Shmem:                80 kB
Slab:              67180 kB
SReclaimable:      57656 kB
SUnreclaim:         9524 kB
KernelStack:        4080 kB
PageTables:         1716 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:     1484252 kB
Committed_AS:     422708 kB
VmallocTotal:   134217728 kB
VmallocUsed:       21452 kB
VmallocChunk:   134191900 kB
HugePages_Total:       0
```

```
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       1024 kB
lnxwas:~ #
```

Table 5-4 lists the field descriptions for `/proc/meminfo`.

Table 5-4 Field descriptions for /proc/meminfo

Field	Description
MemTotal	RAM memory assigned to Linux, not including memory used by the kernel.
MemFree	Memory not currently used by Linux.
Buffers	Memory allocated to file I/O buffers.
Cached	Memory used in the page cache without the amount reported in SwapCached.
SwapCached	Memory that was swapped out and later swapped back in. It continues to reside in swapfile. Pages do not need to be swapped out again if this memory is needed.
Active	Page cache and buffer recently used. This memory is usually not reclaimed for other use.
Inactive	Page cache and buffer cache not recently used. This memory can be reclaimed for other purposes.
HighTotal/High Free	Total and free memory not mapped into the kernel space.
LowTotal/Low Total	Total and free memory directly mapped into the kernel space.
SwapTotal/SwapFree	Swap space available and free swap space.
Dirty	Memory to be written back to permanent media because of changes.
Writeback	Memory actively being written back to permanent media because of changes.
AnonPages	Anonymous memory allocated by the mmap() function and not backed by files.
Mapped	Memory allocated by the mmap() function and backed by files.
Slab	Memory used by the kernel for slab cache to hold data structures.
CommitLimit	Memory that can be allocated on the system if strict overcommit accounting is set (mode 2 in vm.overcommit_ratio). Guarantees that allocated memory will be available when needed.
Committed_AS	Estimated memory required for all processes to be allocated successfully. This memory amount insures that an out of memory situation will not occur. The amount does not reflect memory already in use by all processes but that could be required nonetheless.
PageTables	Memory dedicated to the lowest level of page tables.
VmallocTotal	Memory available to the vmalloc memory area.
VmallocUsed	Memory used in the vmalloc memory area.

Field	Description
VmallocChunk	Contiguous memory available as the vmalloc memory area.
HugePages_Tota	Pool size of huge pages.
HugePages_Free	Number of huge pages not yet allocated in the pool.
HugePages_Rsvp	Number of reserved pages.
HugePages_Surp	Number of surplus huge pages in the pool above the value in /proc/sys/vm/nr_hugepages.

For more details about managing Linux Virtual Memory, see Linux kernel documentation in `/usr/src/linux/Documentation/filesystems/proc.txt` or see this PDF:

http://www.kernel.org/doc/gorman/pdf/understand.pdf

5.6.3 Using the vmstat command

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

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

Delay indicates seconds between updates and *count* indicates the number of updates. Example 5-7 shows the output from the **vmstat** command.

Example 5-7 vmstat output

```
lnxsu2:~ # vmstat 10 7
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0  64140   3420  28572    0    0   307    61   30   73  3  2 94  1  0
 0  5  80408   1624    440   2472  553 8041  1170  8074  333  148  5  3 48 44  0
 0  1  75576  60724    352   6292 6590 5856  7458  5867  377  441  4  3  0 92  0
 0  0   8640  94780    392   7260 3226    0  3323    12  117  209  2  1 80 17  0
```

vmstat results in a display that groups statistics by type (Table 5-5).

Table 5-5 Statistic details from vmstat

Statistic type	Details
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 un-interruptable sleep state
memory	Memory statistics are reported as the average amount of memory: ► swpd: Used memory size (KB) ► free: Unused memory size (KB) ► buff: Bemory allocated to buffers (KB) ► cache: Memory allocated to file system cache (KB)

Statistic type	**Details**
swap	Paging statistics report average paging rates to swap devices: ► si: Paging rate from swap device to memory (KBps) ► so: Paging rate from memory to swap device (KBps)
io	I/O statistics report the average I/O rate to block devices: ► bi : Number of blocks sent to a block device (blocks per second) ► bo: Number of blocks received from a block device (blocks per second)
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: Tme spent in user mode ► sy: Time spent in kernel mode ► id: Time spent idle ► wa: Time spent waiting for I/O ► st: Time spent waiting for a physical CPU, also known as steal time

6

Tuning memory for z/VM Linux guests

In this chapter, we discuss tuning memory and storage for z/VM Linux guests based on z/VM 6.1 and Linux kernel 2.6. We describe new z/VM features, share memory tuning recommendations, discuss how to best use the new Linux kernel technology, and outline general tuning for guest systems.

6.1 What is new in z/VM

z/VM Hypervisor is the most functionally rich virtualization platform available for mainframes. Use it as the foundation hardware whenever you want to exploit IBM virtualization technology on the System z family servers. Linux fits naturally into the z/VM environment, allowing you to run many Linux images on the same LPAR. In fact, some customers use this method to run hundreds of Linux images in production mode. You can create virtual servers on demand, reducing the time to deploy new servers with the powerful build and clone capability, especially when consolidating from distributed platforms onto System z. This is also a unique advantage for development groups who need to build test environments quickly to meet project deadlines.

6.1.1 z/VM 5.3

Upgrades and features to z/VM 5.3 offer constraint relief for large, memory-intensive, real-memory virtual server environments. These are some of those features:

- Extended scalability to allow 256 GB of real memory, 8 TB total of virtual storage, and 32 real processors in a single z/VM image.
- Added support for the Collaborative Memory Management Assist (CMMA) and Virtual Machine Resource Manager (VMRM). Both CMMA and VMRM detect when memory is constrained and permit the Linux guests to adjust memory consumption to relieve the memory constraint.
- Removed many memory contention issues existing in previous releases. Allowed the Control Program (CP) to use memory higher than 2 GB. Previously, guest pages had to be moved below 2 GB for various reasons, often in standard I/O and Queued Direct I/O (QDIO). Now I/O can use buffers anywhere in real memory. QDIO structures can reside above 2 GB, along with most CP control blocks.

6.1.2 z/VM 5.4

z/VM 5.4 provides major improvements for System z servers with large memory configurations. This release improves scalability, supports increased server workloads, and exploits new System z capabilities, such as these:

- Greater flexibility that supports the new z/VM-mode partitions, which allows all System z processor types (CPs, IFLs, zIIPs, zAAPs, and ICFs) to be defined in the same z/VM LPAR for guest operating systems.
- Includes the z/VM HyperSwap® function to provide a coordinated, near continuous availability and disaster recovery for distributed applications, such as WebSphere. z/VM HyperSwap spans all Linux guests running under z/VM.
- Dynamic memory upgrade support that allows you to add real memory to a running z/VM system without shutting down z/VM, guests, and the corresponding tasks that are related to LPAR memory activation. Begin adding memory by defining it as *standby* for LPAR. This allows z/VM to reconfigure memory dynamically and use it immediately. You can also use this method with individual guests that support the dynamic memory reconfiguration architecture.
- Allows you to install Linux on System z from the HMC, which eliminates the need for a network setup or an LPAR and HMC connection.
- Exploits all OSA-Express3 ports with enhanced physical connectivity, which also services the network and reduces the number of required resources.

To learn more about z/VM 5.4, use this link:

`http://www.vm.ibm.com/zvm540/`

6.1.3 z/VM 6.1

z/VM 6.1 will be the baseline for all future z/VM enhancements. This release uses the new Architecture Level Set (ALS) that is available only on IBM System z10 and zEnterprise™ servers. ALS allows z/VM to take full advantage of new hardware technology as it becomes available.

These are some z/VM V6.1 enhancements:

- Dynamically configures processors, channels, I/O, and memory for both the z/VM system and guests. This reduces the need to re-IPL z/VM
- Provides enhanced memory use between z/VM and Linux servers with the Virtual Machine Resource Manager (VMRM)
- Defines Discontiguous Saved Segments (DCSS) above 2047 MB in virtual storage, thus allowing multiple DCSSs to be used together
- Enhances performance of virtual networking environments that run heavy guest-to-guest streaming workloads
- Enhances z/VM internal processing scalability in the memory management subsystem, for example:
  - Configurable CP settings to dynamically change the workings of page reorder processing
  - Improved page release serialization to reduce the amount of resource required to manipulate guest pages
  - Enhanced contiguous frame coalescing to allow the hypervisor to better serve multiple contiguous pages to a guest virtual machine
- Faster data access for FICON® Express8
- Closer integration with IBM Systems Director, which eliminates the need to download agents and simplifies agent installation
- Significantly better and more secure guest transactions for Crypto Express3 than for Crypto Express2
- Guest support for IBM System Storage® DS8000 Extended Address Volumes (EAVs) to simplify storage management and to relieve address constraints

For more details about z/VM 6.1, see this website:

`http://www.vm.ibm.com/zvm610`

6.1.4 z/VM 6.2

This section provides an overview of the new functions, enhancements, and changes included in z/VM V6.2, in addition to extensions to z/VM virtualization technology that support Linux on System z, z/OS, and other guests.

- SSI Cluster

  z/VM V6.2 implements multisystem virtualization using a z/VM single system image (SSI) cluster composed of up to four z/VM systems. This mainframe multisystem virtualization technology extends the z/VM virtualization technology to a new level that allows cluster members to share resources and synchronize between nodes while appearing to be a single system.

  z/VM SSI cluster members are part of the same Inter-System Facility for Communications (ISFC) collection and communicate with ISFC channel connections. Cluster members also share DASD for virtual machines and selected z/VM data, as well as LAN segments and IP subnets. The concept of a global virtual switch provides identical network connectivity across all active members within a cluster.

  z/VM instances that are members of an SSI cluster can be serviced and administered as one system, which simplifies z/VM systems management. Resources used by both CP and virtual machines are coordinated and shared with all members. This allows Linux guests to access the same devices and networks regardless of which member they are logged on to or relocated to. Shared resources include:

  – User directory
  – Minidisks
  – Spool files
  – Network device MAC addresses

  Each z/VM SSI cluster member can communicate with all other active members. When a z/VM system is configured as a member of a cluster, it automatically joins the other members during system startup. The system seamlessly coordinates members joining and leaving the cluster, maintains a common view of member and resource states, and negotiates access to shared cluster resources.

- Live guest relocation

  Live guest relocation (LGR) allows you to move a running Linux virtual machine from one z/VM system to another within a z/VM SSI cluster without disrupting the system. LGR provides continuity for virtual server workloads during planned z/VM and machine outages (for example, service and hardware upgrades) by allowing applications to be available during the outages with less impact to the application and with less setup. The system verifies that resources and features are available on the destination system before relocating the virtual machine.

  You can also request a verification to assess whether a guest can be relocated. In an SSI cluster of different machine models, the architecture level for each guest is tailored to the features set common to the member systems in the guest's relocation domain. It is important to identify the Linux level in each guest, as unsupported Linux levels might not operate correctly after relocation.

z/VM V6.2 supports IBM zEnterprise 196 (z196) and IBM zEnterprise 114 (z114). For more details, see this website:

`http://www.vm.ibm.com`

6.2 Memory tuning recommendations

Storage recommendations are useful in virtualized environments, especially because the Hypervisor can overcommit resources. Even though z/VM is the most functionally capable Hypervisor operating system available today, we recommend that you fine tune key resources (for example, memory) for best system performance.

6.2.1 VM storage management overview

The z/VM system maps the guest virtual memory into the System z machine real memory storage. Active guests' virtual pages are moved to XSTOR if there are not enough real memory frames to contain all of active guests. If XSTOR is full, the guests' pages are moved to the DASD paging space. Also, CP provides expanded storage as a minidisk cache that shares expanded storage with paging.

z/VM manages a paging hierarchy that uses expanded storage first, then slower DASD (Figure 6-1). Pages move to the slower paging DASD when not referenced within a given time limit. Pages that are referenced within the time limit are quickly brought back into central storage. This paging design gives users a consistent response.

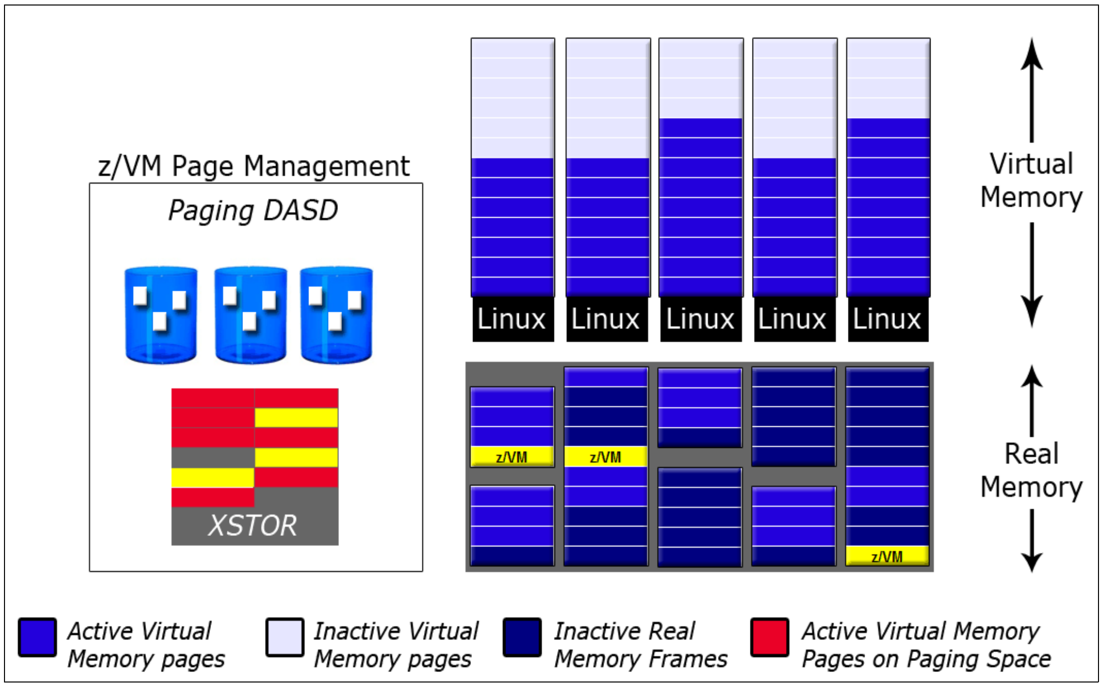

z/VM Page Management
Paging DASD
XSTOR
Linux
Linux
Linux
Linux
Linux
z/VM
z/VM
z/VM
Virtual Memory
Real Memory
Active Virtual Memory pages
Inactive Virtual Memory pages
Inactive Real Memory Frames
Active Virtual Memory Pages on Paging Space

Figure 6-1 VM storage management overview

z/VM treats expanded storage as another level of real storage, and whenever possible, all paging goes to expanded storage first. When paging use exceeds a high threshold, old pages from expanded storage are migrated to DASD. Direct paging to DASD occurs only if expanded storage is full and migrating to create room in expanded storage cannot keep up with the paging demand.

6.2.2 Reducing operational machine size

We recommend that you define the virtual machine to a small size, depending on the applications and workloads that you need to run. Certain Linux distributions can have minimum requirements as low as 64 MB. Certain applications run well in such configurations, whereas others might require larger virtual machines.

Do not define a virtual machine size larger than necessary because Linux will use the excess memory for file and buffer caches. On a standalone system, these buffers can help certain workloads to avoid I/O. However, in a virtual environment, the extra memory consumed adds to overall system memory contention. This could cause a negative impact greater than the positive impact of avoiding I/O, which is especially true in configurations where data is mostly read and is shared heavily between guests. z/VM minidisk caching can help avoid I/O in such configurations.

The working set size (WSS) of a virtual machine is a projection of the number of pages allocated in real storage necessary to efficiently process transactions with a minimum of page faults. WSS is based on the run history and is calculated each time that the virtual machine is dropped from the dispatch list. Virtual machine pages that are not in real storage are located in paging storage, either in expanded storage or on paging devices (DASD).

As a general guideline, define the virtual machine memory size to keep Linux on the verge of swapping. Lower the memory size until you see Linux begin to swap, then increase the virtual machine memory to the next larger size.

Tips to reduce operational memory size

Minimize server memory footprint. Use any of these methods to reduce operational memory size:

- Eliminate unneeded processes.

  Remember to do this even though it sounds obvious. Monitor servers that are running all their application and middleware software but that are not executing transactions, only background processes, such as cron, then eliminate any unnecessary processes.

- Divide large servers into smaller specialized servers.

  You can explicitly tune a service machine to perform a certain function. Some Linux servers running many applications do not have these controls, but here you can separate functions into smaller Linux guests. We do not recommend that you separate every server, because a separated guest needs a certain amount of memory to run its Linux operating system.

- Reduce machine size.

  Minimize the virtual machine size to the lowest possible amount, but only after researching and testing your system. Read more about sizing of the Linux guest in 5.5, “Sizing Linux virtual memory” on page 60.

- Use a virtual disk for a swap device.

  Use the DIAGNOSE access method on a VDISK as a swap device to reduce the swapping performance penalty. This is applicable only if you have a large amount of real memory available on System z. By using virtual disk for swap, you reduce the I/O generated by swapping on real device while increasing the total amount of storage used. This method is useful when Linux and z/VM generate a high paging rate, a situation that might create double paging (that is, z/VM pages out guest storage already swapped out by Linux). Using VDISK as a swap device will reduce the effects of double paging.

Figure 6-2 shows typical VDISK storage usage.

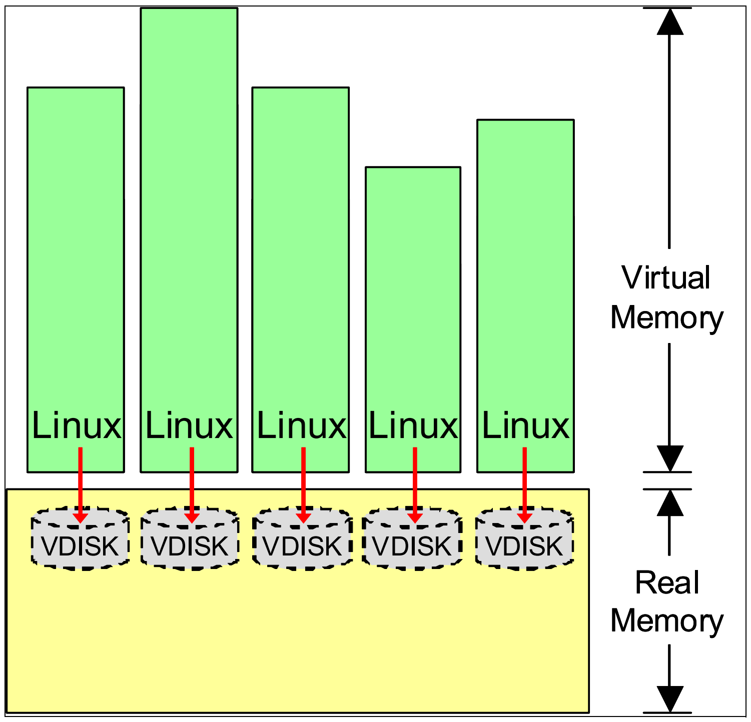


Figure 6-2 VDISK usage

6.2.3 Reducing infrastructure storage costs

We recommend that you take all opportunities to reduce your infrastructure storage costs, including the following:

- Use the DIAGNOSE driver for DASD and MDC record cache.

  You can use the DIAGNOSE driver to reduce MDC required storage by using a record-level mini cache. To learn about the DIAGNOSE driver, see *IBM Linux on System z - Device Drivers, Features, and Commands*, SC33-8289.

- Use CMM.

  VMRM Cooperative Memory Management (VMRM-CMM) between z/VM and Linux guests manages memory constraint in the system. VMRM uses variables obtained from system and storage domain CP monitor data to detect constraint, and notifies specific Linux virtual guests. The guests can then adjust their memory utilization to relieve the constraint. For more details, see *z/VM Performance*, SC24-6208.

- Use shared kernel or execute-in-place technology.

  You can use NSS and DCSS to reduce the total amount of storage needed to manage Linux instances. We recommend that you review and evaluate each environment to determine whether this solution can be effective or worthwhile. See 6.7, “Execute-in-place technology” on page 95, for more information.

6.3 Enhancing storage

Periodically take steps to control storage subsystems. Effectively managing contiguous frames can reduce memory management overhead and improve overall performance. Skillfully managing real memory will also help you handle memory constraints. The better that you manage memory, the less you need to spread large workloads across multiple z/VM LPARs.

The amount of usable real and virtual memory depends on the amount of real memory in the z/VM logical partition, the hardware server model, firmware level, and configuration, and the number of guests and their workload characteristics. Since release z/VM V5.3, z/VM can support significantly more real memory, up to 256 GB, actually twice the size supported by previous releases, and more than 1 terabyte (TB) of total virtual memory for guests.

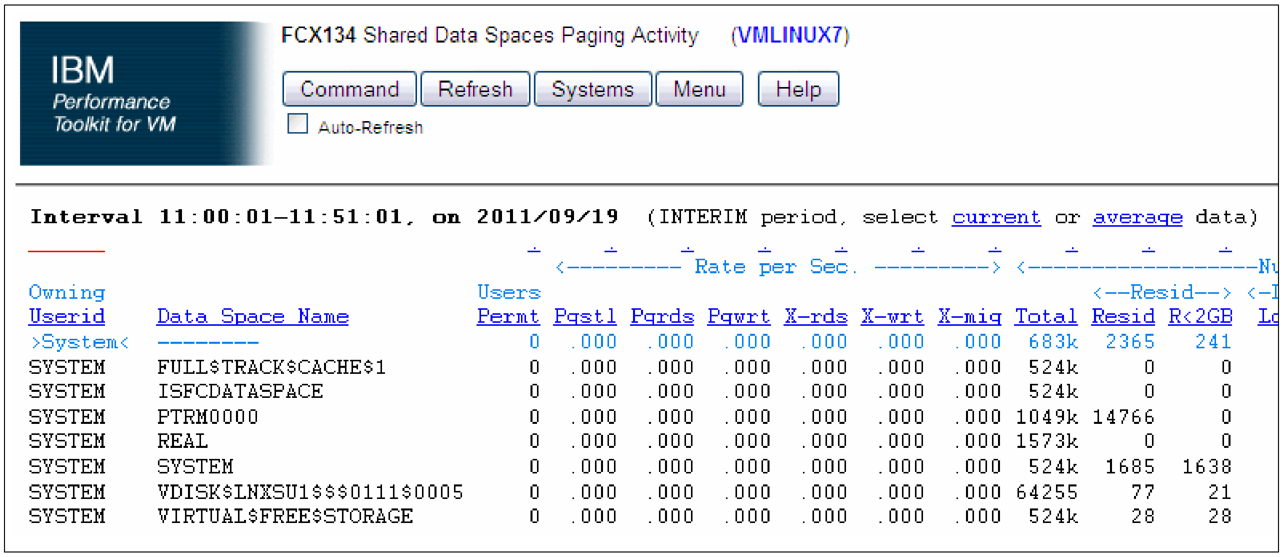
z/VM 6.1 includes important enhancements to CP storage management. This release eliminates many of the memory contention issues of previous releases and allows CP to use memory above the 2 GB line. Before z/VM 6.1, guest pages often had to be moved below the 2 GB (for example, in both standard I/O and QDIO). Now you can use buffers anywhere in real memory for I/O and QDIO structures, and most CP control blocks can reside around 2 GB. These improvements provide constraint relief for large memory-intensive virtual server environments. Learn more by reading the z/VM Performance Report available here:

http://www.vm.ibm.com/perf/reports/zvm/html/index.html

6.3.1 Increasing maximum in-use virtual storage

CP has to create an 8 KB mapping structure, PaGe Management BlocK (PGMBK), before any page can be referenced in a 1-megabyte segment of virtual storage. z/VM 6.1 supports larger amounts of in-use virtual storage because this page management block now resides above the 2 GB line.

You can verify that z/VM 6.1 is not using PGMBK under the 2 GB line in the Shared Data Spaces Paging Activity (DSPACESH) report. Use the Performance Toolkit, panel FCX for the report (Figure 6-3).



FCX134 Shared Data Spaces Paging Activity (VMLINUX7)

```
Interval 11:00:01-11:51:01, on 2011/09/19  (INTERIM period, select current or average data)

                                           <---------- Rate per Sec. ----------> <-------------------Nu
Owning                                   Users                                                  <--Resid--> <-I
Userid      Data Space Name              Permt  Pgstl  Pgrds  Pgwrt  X-rds  X-wrt  X-mig  Total  Resid R<2GB  Lo
>System<    ---------                        0   .000   .000   .000   .000   .000   .000   683k   2365   241
SYSTEM      FULL$TRACK$CACHE$1               0   .000   .000   .000   .000   .000   .000   524k      0     0
SYSTEM      ISFCDATASPACE                    0   .000   .000   .000   .000   .000   .000   524k      0     0
SYSTEM      PTRM0000                         0   .000   .000   .000   .000   .000   .000  1049k  14766     0
SYSTEM      REAL                             0   .000   .000   .000   .000   .000   .000  1573k      0     0
SYSTEM      SYSTEM                           0   .000   .000   .000   .000   .000   .000   524k   1685  1638
SYSTEM      VDISK$LNXSU1$$$0111$0005         0   .000   .000   .000   .000   .000   .000  64255     77    21
SYSTEM      VIRTUAL$FREE$STORAGE             0   .000   .000   .000   .000   .000   .000   524k     28    28
```

Figure 6-3 FCX134 - Shared Data Spaces Paging Activity (DSPACESH)

6.3.2 z/VM reorder processing

Page reordering is the z/VM process of managing user frame owned lists for Demand Scan processing. z/VM tracks real memory frames associated with each guest virtual machine. The resulting list is used to find memory frames with different characteristics during a Demand Scan, which is triggered when z/VM detects that there are insufficient frames on its available list. While the reorder processing is running, there is a serialization in the guest even if you assigned more than one virtual processor to the guest. During the reorder process, the guest machine will appear to be halted. Here are key points about reorder processing:

- It resets the HW reference bit.
- It serializes the virtual machine (that is, all virtual processors).
- It is done periodically on a virtual machine basis.
- The cost of reorder is proportional to the number of resident frames for a virtual machine.

Use the link below to learn more about reorder processing:

`http://www.vm.ibm.com/perf/tips/reorder.html`

6.4 Investigating performance issues in guests

Performance issues are not isolated problems but are usually related to an element of the guest environment. You need to evaluate key performance indicators to fully understand performance issues and resolve them correctly. This section discusses key performance indicators that can help you solve performance issues.

6.4.1 Key performance indicators

Key performance indicators in Performance Toolkit reports will help you understand your system behavior and to better respond to performance issues. A good practice is to closely monitor both your z/VM system and workload processing to identify possible performance issues before they affect guests. Knowing your environment helps you to determine which factors will result in a performance gain and which could cause bottlenecks. You should also routinely check the guest machine memory performance indicators.

6.4.2 General z/VM system performance indicators

There are two main performance areas to consider in a z/VM environment:

- The performance of the total system
- The performance of individual virtual machines

Processor usage is an important factor in determining possible performance. Another important factor is the interconnection of the CPU, storage, paging, and I/O.

6.4.3 Performance Toolkit reports

The Performance Toolkit generates reports that can help you analyze performance indicators in your system. We describe some reports here. For report details and for additional reports, see the z/VM Performance Toolkit Guide found at:

`http://publibz.boulder.ibm.com/epubs/book/hcsm7c01.boo`

Also see the Performance Toolkit Reference found at:

`http://publibz.boulder.ibm.com/epubs/book/hcsl7c01.boo`

Total processor utilization report

Tookit report: FCX100 CPU

Figure 6-4 on page 79 shows the output of the Performance Toolkit report FCX100 CPU. Keep these definitions in mind as you use the reports:

- System Time

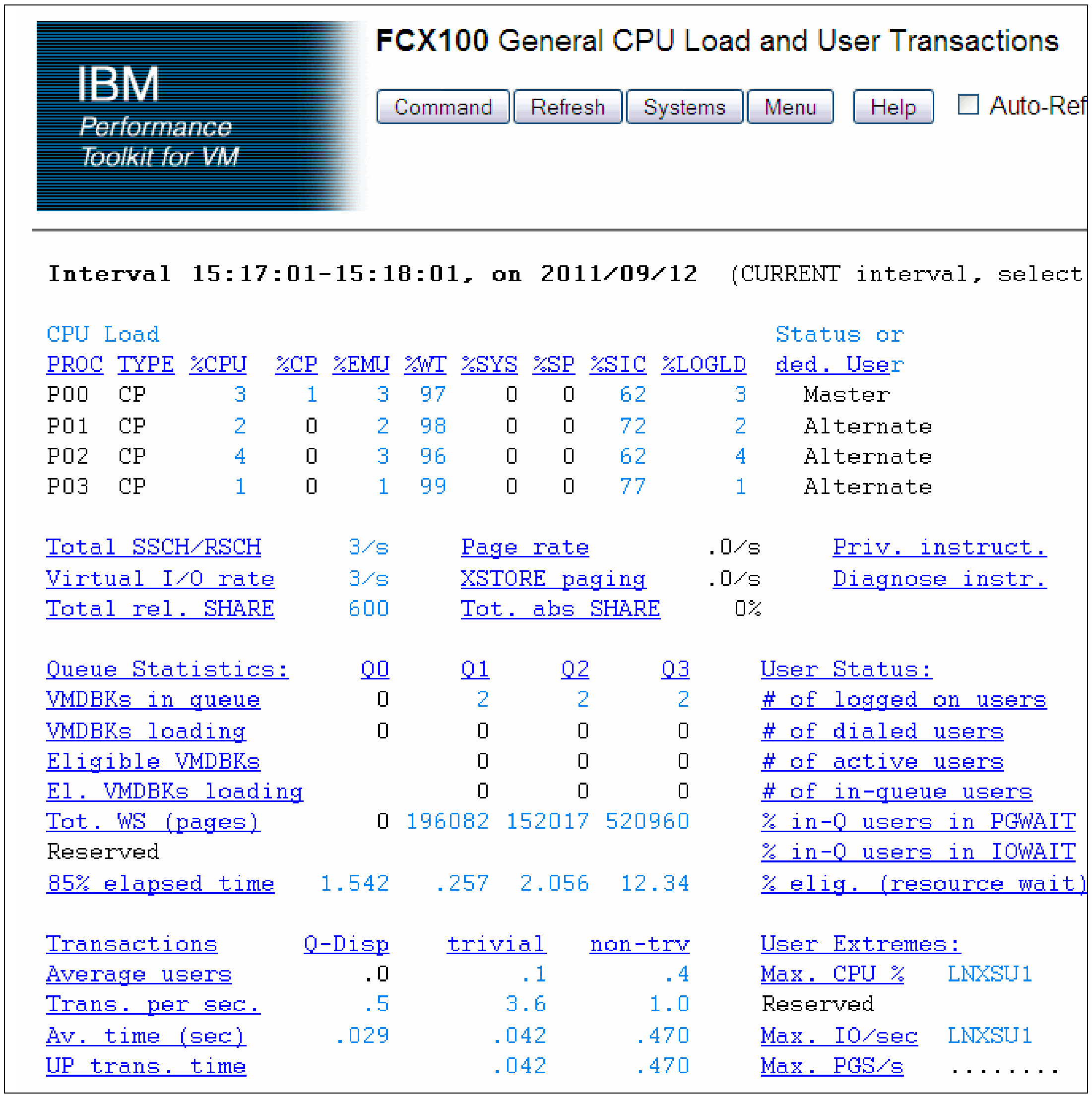
  The processor time used by the CP for system functions that are not directly related to any virtual machine. This should be less than 10% of the total processor time of the z/VM LPAR.

- CP Processor Time

  The processor time used by the CP in support of individual virtual machines.

- Virtual Processor Time
- (Emulation Mode)

  The processor time consumed by the virtual machine and the applications in it.



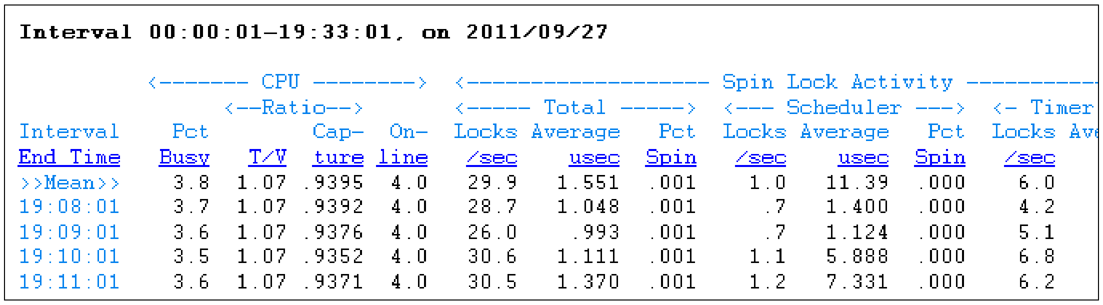
```
FCX100 General CPU Load and User Transactions

Interval 15:17:01-15:18:01, on 2011/09/12  (CURRENT interval, select

CPU Load                                              Status or
PROC TYPE %CPU   %CP %EMU %WT %SYS %SP %SIC %LOGLD    ded. User
P00  CP      3     1    3  97    0   0   62      3     Master
P01  CP      2     0    2  98    0   0   72      2     Alternate
P02  CP      4     0    3  96    0   0   62      4     Alternate
P03  CP      1     0    1  99    0   0   77      1     Alternate

Total SSCH/RSCH      3/s     Page rate         .0/s     Priv. instruct.
Virtual I/O rate     3/s     XSTORE paging     .0/s     Diagnose instr.
Total rel. SHARE     600     Tot. abs SHARE      0%

Queue Statistics:      Q0      Q1      Q2      Q3     User Status:
VMDBKs in queue         0       2       2       2     # of logged on users
VMDBKs loading          0       0       0       0     # of dialed users
Eligible VMDBKs                 0       0       0     # of active users
El. VMDBKs loading              0       0       0     # of in-queue users
Tot. WS (pages)         0  196082  152017  520960     % in-Q users in PGWAIT
Reserved                                              % in-Q users in IOWAIT
85% elapsed time    1.542    .257   2.056   12.34     % elig. (resource wait)

Transactions       Q-Disp   trivial   non-trv     User Extremes:
Average users          .0        .1        .4     Max. CPU %    LNXSU1
Trans. per sec.        .5       3.6       1.0     Reserved
Av. time (sec)       .029      .042      .470     Max. IO/sec   LNXSU1
UP trans. time                 .042      .470     Max. PGS/s    ........
```

Figure 6-4 FCX100 CPU report output

Total to Virtual Ratio

Tookit report: FCX239 Processor Performance Summary by Time (PROCSUM)

Figure 6-5 shows the output of this report. z/VM efficiency is better the closer that the ratio is to 1.0. Check the system if the ratio is over 1.30.



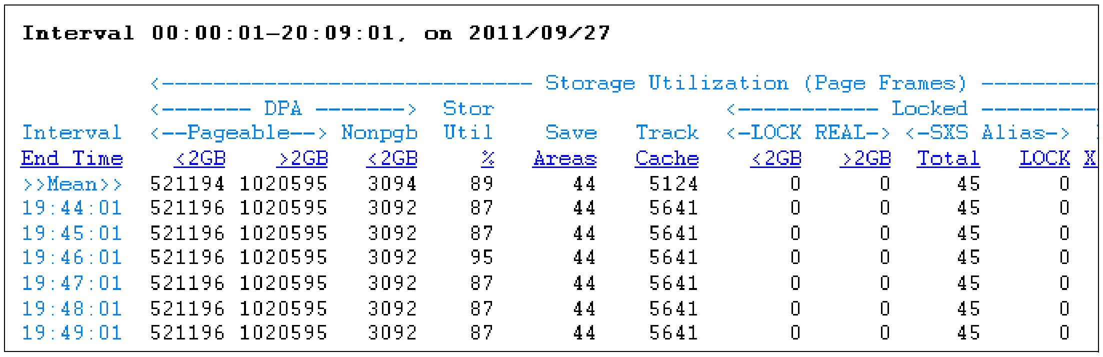
```
Interval 00:00:01-19:33:01, on 2011/09/27

          <-------- CPU ---------> <-------------------- Spin Lock Activity -----------
                <--Ratio-->        <----- Total -----> <--- Scheduler ---> <- Timer
Interval   Pct        Cap-  On-    Locks Average   Pct Locks Average   Pct Locks Ave
End Time   Busy  T/V  ture  line   /sec     usec  Spin  /sec    usec  Spin  /sec
>>Mean>>   3.8  1.07 .9395  4.0   29.9    1.551  .001   1.0   11.39  .000   6.0
19:08:01   3.7  1.07 .9392  4.0   28.7    1.048  .001    .7   1.400  .000   4.2
19:09:01   3.6  1.07 .9376  4.0   26.0     .993  .001    .7   1.124  .000   5.1
19:10:01   3.5  1.07 .9352  4.0   30.6    1.111  .001   1.1   5.888  .000   6.8
19:11:01   3.6  1.07 .9371  4.0   30.5    1.370  .001   1.2   7.331  .000   6.2
```

Figure 6-5 FCX239 PROCSUM

Pageable Real Memory

Tookit report: FCX253 STORLOG and FCX143 PAGELOG

Figure 6-6 displays output from this report. Pageable real memory is what remains when the CP nucleus and non-pageable control blocks are taken out of the total real memory.



```
Interval 00:00:01-20:09:01, on 2011/09/27

          <------------------------------ Storage Utilization (Page Frames) ---------
          <------- DPA -------->  Stor          <------------ Locked ----------
Interval  <--Pageable--> Nonpgb  Util    Save   Track  <-LOCK REAL-> <-SXS Alias->
End Time    <2GB    >2GB   <2GB     %   Areas   Cache    <2GB   >2GB   Total   LOCK X
>>Mean>>  521194 1020595   3094    89      44    5124       0      0      45      0
19:44:01  521196 1020595   3092    87      44    5641       0      0      45      0
19:45:01  521196 1020595   3092    87      44    5641       0      0      45      0
19:46:01  521196 1020595   3092    95      44    5641       0      0      45      0
19:47:01  521196 1020595   3092    87      44    5641       0      0      45      0
19:48:01  521196 1020595   3092    87      44    5641       0      0      45      0
19:49:01  521196 1020595   3092    87      44    5641       0      0      45      0
```

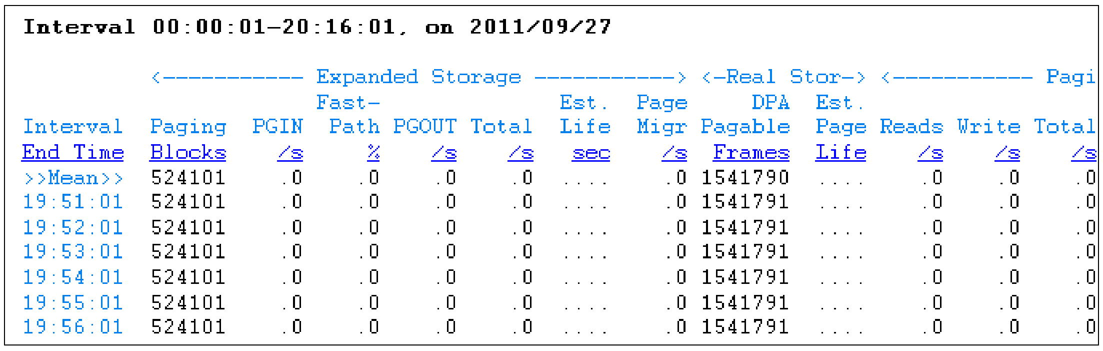
Figure 6-6 FCX253 STORLOG

Paging I/O to DASD

Tookit report: FCX143 PAGELOG

Figure 6-7 shows the Pagelog report output. Note these items regarding this report:

PGIN	Page rate of operations from central storage to expanded storage.
PGOUT	Page rate of operations from expanded storage to central storage.
MIGRATION Rate	Rate of page movement from expanded storage out to DASD.



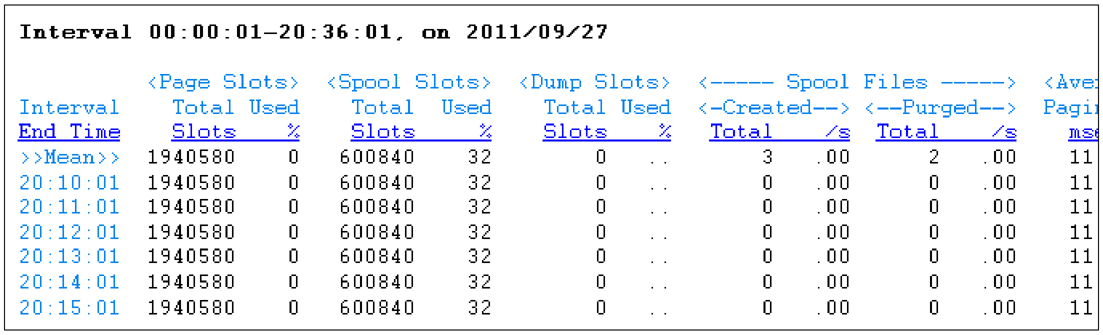
```
Interval 00:00:01-20:16:01, on 2011/09/27

          <----------- Expanded Storage -----------> <-Real Stor-> <----------- Pagi
                         Fast-             Est.  Page     DPA  Est.
Interval  Paging  PGIN  Path PGOUT Total   Life  Migr Pagable  Page Reads Write Total
End Time  Blocks    /s     %    /s    /s    sec    /s  Frames  Life    /s    /s    /s
>>Mean>>  524101    .0    .0    .0    .0   ....    .0 1541790  ....    .0    .0    .0
19:51:01  524101    .0    .0    .0    .0   ....    .0 1541791  ....    .0    .0    .0
19:52:01  524101    .0    .0    .0    .0   ....    .0 1541791  ....    .0    .0    .0
19:53:01  524101    .0    .0    .0    .0   ....    .0 1541791  ....    .0    .0    .0
19:54:01  524101    .0    .0    .0    .0   ....    .0 1541791  ....    .0    .0    .0
19:55:01  524101    .0    .0    .0    .0   ....    .0 1541791  ....    .0    .0    .0
19:56:01  524101    .0    .0    .0    .0   ....    .0 1541791  ....    .0    .0    .0
```

Figure 6-7 FCX143 PAGELOG

Percentage Allocation of Page DASD space

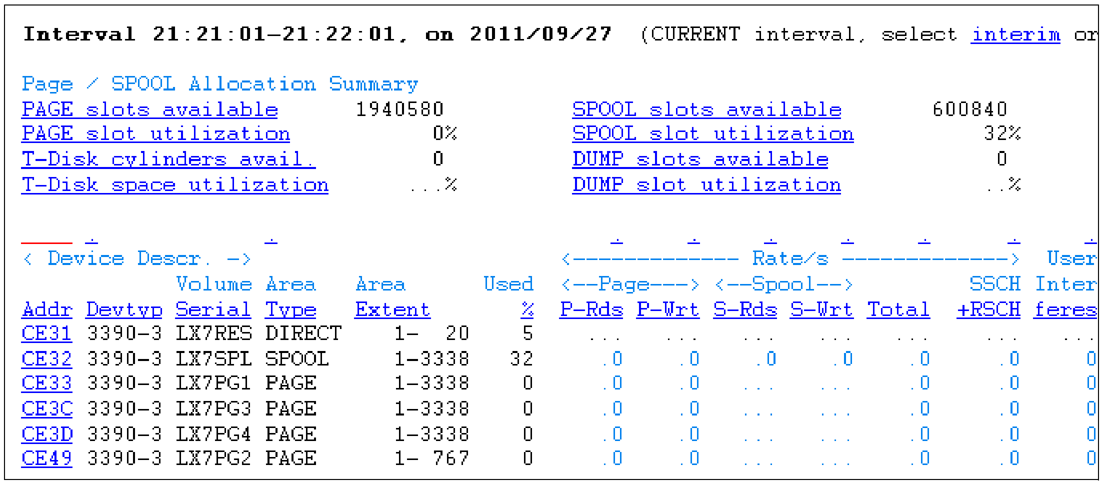
Tookit report: FCX146 AUXLOG and FCX109 DEV CPOWN

Figure 6-8 and Figure 6-9 show the output of this report. Note that the amount of paging DASD should not be higher than 50%.



```
Interval 00:00:01-20:36:01, on 2011/09/27

          <Page Slots>  <Spool Slots>  <Dump Slots>  <----- Spool Files -----> <Ave
Interval   Total Used    Total  Used    Total Used  <-Created--> <--Purged--> Pagi
End Time   Slots    %    Slots     %    Slots    %  Total     /s Total     /s   ms
>>Mean>>  1940580   0   600840    32        0   ..      3    .00     2    .00   11
20:10:01  1940580   0   600840    32        0   ..      0    .00     0    .00   11
20:11:01  1940580   0   600840    32        0   ..      0    .00     0    .00   11
20:12:01  1940580   0   600840    32        0   ..      0    .00     0    .00   11
20:13:01  1940580   0   600840    32        0   ..      0    .00     0    .00   11
20:14:01  1940580   0   600840    32        0   ..      0    .00     0    .00   11
20:15:01  1940580   0   600840    32        0   ..      0    .00     0    .00   11
```

Figure 6-8 FCX146 AUXLOG



```
Interval 21:21:01-21:22:01, on 2011/09/27  (CURRENT interval, select interim or

Page / SPOOL Allocation Summary
PAGE slots available       1940580          SPOOL slots available        600840
PAGE slot utilization           0%          SPOOL slot utilization          32%
T-Disk cylinders avail.         0           DUMP slots available            0
T-Disk space utilization      ...%          DUMP slot utilization         ..%


< Device Descr. ->                        <-------------- Rate/s -------------->  User
             Volume Area    Area     Used <--Page---> <--Spool-->          SSCH  Inter
Addr Devtyp Serial Type   Extent      %  P-Rds P-Wrt S-Rds S-Wrt Total  +RSCH feres
CE31 3390-3 LX7RES DIRECT    1-   20   5    ...   ...   ...   ...   ...    ...   ...
CE32 3390-3 LX7SPL SPOOL     1-3338   32     .0    .0    .0    .0    .0     .0     0
CE33 3390-3 LX7PG1 PAGE      1-3338    0     .0    .0   ...   ...    .0     .0     0
CE3C 3390-3 LX7PG3 PAGE      1-3338    0     .0    .0   ...   ...    .0     .0     0
CE3D 3390-3 LX7PG4 PAGE      1-3338    0     .0    .0   ...   ...    .0     .0     0
CE49 3390-3 LX7PG2 PAGE      1- 767    0     .0    .0   ...   ...    .0     .0     0
```

Figure 6-9 DEV CPOWN

6.4.4 Guest virtual machine performance indicators

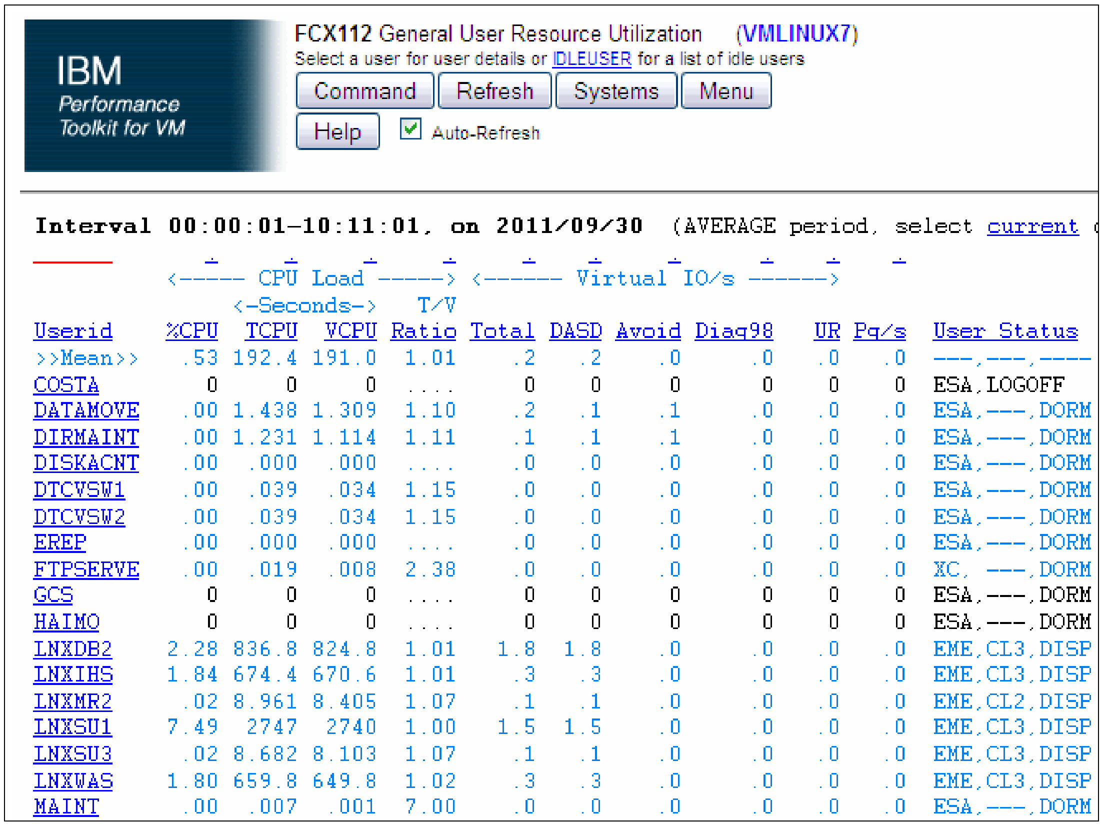
The Performance Toolkit also provides reports for guest virtual machines. This section reviews some of the main reports.

General user utilization

Toolkit report: FCX112 General User Resource Utilization (USER)

The FCX112 report outlines the entire resource utilization of virtual machines in z/VM system (Figure 6-10). Report data is grouped into these blocks:

- ► The first block always shows the same layout for all users, along with general performance information for that user. Fields that show user activity are highlighted.
- ► The second block is for virtual MP users only and displays CPU and I/O load for each virtual processor along with its corresponding status.
- ► Another block shows detailed information about the paging activity for each of the data spaces owned by the user.



FCX112 General User Resource Utilization (VMLINUX7)

Select a user for user details or IDLEUSER for a list of idle users

```
Interval 00:00:01-10:11:01, on 2011/09/30  (AVERAGE period, select current 
                 <----- CPU Load -----> <------ Virtual IO/s ------>
                      <-Seconds->   T/V
Userid     %CPU  TCPU  VCPU Ratio Total  DASD Avoid Diag98   UR  Pq/s  User Status
>>Mean>>    .53 192.4 191.0  1.01    .2    .2    .0     .0   .0    .0  ---,---,----
COSTA         0     0     0  ....     0     0     0      0    0     0  ESA,LOGOFF
DATAMOVE    .00 1.438 1.309  1.10    .2    .1    .1     .0   .0    .0  ESA,---,DORM
DIRMAINT    .00 1.231 1.114  1.11    .1    .1    .1     .0   .0    .0  ESA,---,DORM
DISKACNT    .00  .000  .000  ....    .0    .0    .0     .0   .0    .0  ESA,---,DORM
DTCVSW1     .00  .039  .034  1.15    .0    .0    .0     .0   .0    .0  ESA,---,DORM
DTCVSW2     .00  .039  .034  1.15    .0    .0    .0     .0   .0    .0  ESA,---,DORM
EREP        .00  .000  .000  ....    .0    .0    .0     .0   .0    .0  ESA,---,DORM
FTPSERVE    .00  .019  .008  2.38    .0    .0    .0     .0   .0    .0  XC, ---,DORM
GCS           0     0     0  ....     0     0     0      0    0     0  ESA,---,DORM
HAIMO         0     0     0  ....     0     0     0      0    0     0  ESA,---,DORM
LNXDB2     2.28 836.8 824.8  1.01   1.8   1.8    .0     .0   .0    .0  EME,CL3,DISP
LNXIHS     1.84 674.4 670.6  1.01    .3    .3    .0     .0   .0    .0  EME,CL3,DISP
LNXMR2      .02 8.961 8.405  1.07    .1    .1    .0     .0   .0    .0  EME,CL2,DISP
LNXSU1     7.49  2747  2740  1.00   1.5   1.5    .0     .0   .0    .0  EME,CL3,DISP
LNXSU3      .02 8.682 8.103  1.07    .1    .1    .0     .0   .0    .0  EME,CL3,DISP
LNXWAS     1.80 659.8 649.8  1.02    .3    .3    .0     .0   .0    .0  EME,CL3,DISP
MAINT       .00  .007  .001  7.00    .0    .0    .0     .0   .0    .0  ESA,---,DORM
```

Figure 6-10 FCX112 General User Resource Utilization (USER)

Here are descriptions for some of the main fields in this report:

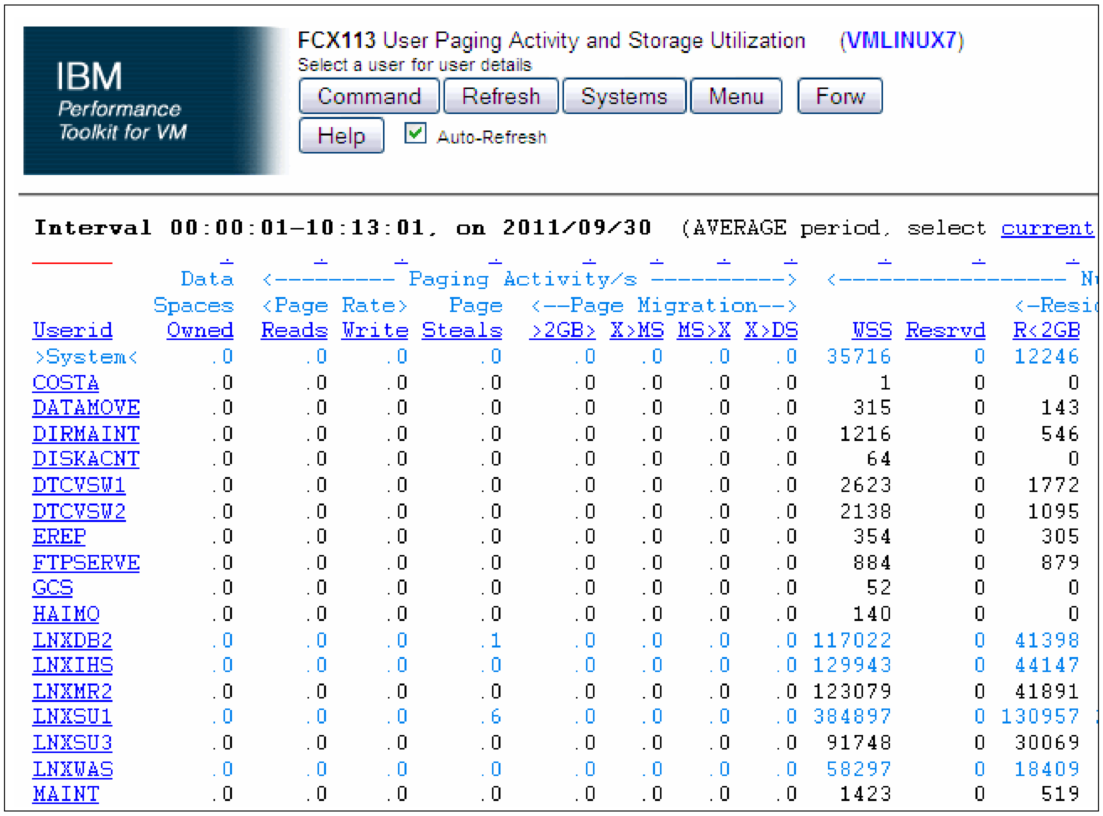
CPU%	Percent of total CPU used, based on the utilization of a single processor. Values exceeding 100% are possible for virtual MP users.
TCPU	Seconds of total CPU time used during the interval.
VCPU	Seconds of virtual CPU time used during the interval (emulation time).
T/V Ratio	Ratio of total to virtual (emulation) CPU time used.
Virtual I/O	Total virtual I/O rate per second in the selected interval.

Memory Distribution

Tookit report: FCX113 UPAGE

The FCS113 UPAGE report shows the virtual machine memory utilization usage (Figure 6-11). Here are the descriptions for the main fields shown in this report:

Resident	Amount of virtual machine memory residing in z/VM real memory.
Expanded storage	Amount of virtual machine memory out to expanded storage.
DASD slots	Amount of virtual machine memory paged out to DASD.
Guest Page Reads	Page rate from DASD to central storage.
Guest Page Writes	Page rate from central storage to DASD.
Guest PGIN Rate	Page rate from expanded storage to central storage.
Guest PGOUT Rate	Page rate from central storage to expanded storage.
Guest Migrate Rate	Page rate of migration from expanded storage to DASD.


IBM
Performance Toolkit for VM
FCX113 User Paging Activity and Storage Utilization (VMLINUX7)
Select a user for user details
Command
Refresh
Systems
Menu
Forw
Help
Auto-Refresh

```
Interval 00:00:01-10:13:01, on 2011/09/30  (AVERAGE period, select current

             Data  <---------- Paging Activity/s ---------->  <------------------ N
           Spaces  <Page Rate>   Page  <--Page Migration-->                 <-Resi
Userid      Owned  Reads Write Steals  >2GB> X>MS  MS>X X>DS     WSS Resrvd  R<2GB
>System<       .0     .0    .0     .0     .0   .0    .0   .0   35716      0  12246
COSTA          .0     .0    .0     .0     .0   .0    .0   .0       1      0      0
DATAMOVE       .0     .0    .0     .0     .0   .0    .0   .0     315      0    143
DIRMAINT       .0     .0    .0     .0     .0   .0    .0   .0    1216      0    546
DISKACNT       .0     .0    .0     .0     .0   .0    .0   .0      64      0      0
DTCVSW1        .0     .0    .0     .0     .0   .0    .0   .0    2623      0   1772
DTCVSW2        .0     .0    .0     .0     .0   .0    .0   .0    2138      0   1095
EREP           .0     .0    .0     .0     .0   .0    .0   .0     354      0    305
FTPSERVE       .0     .0    .0     .0     .0   .0    .0   .0     884      0    879
GCS            .0     .0    .0     .0     .0   .0    .0   .0      52      0      0
HAIMO          .0     .0    .0     .0     .0   .0    .0   .0     140      0      0
LNXDB2         .0     .0    .0     .1     .0   .0    .0   .0  117022      0  41398
LNXIHS         .0     .0    .0     .0     .0   .0    .0   .0  129943      0  44147
LNXMR2         .0     .0    .0     .0     .0   .0    .0   .0  123079      0  41891
LNXSU1         .0     .0    .0     .6     .0   .0    .0   .0  384897      0 130957
LNXSU3         .0     .0    .0     .0     .0   .0    .0   .0   91748      0  30069
LNXWAS         .0     .0    .0     .0     .0   .0    .0   .0   58297      0  18409
MAINT          .0     .0    .0     .0     .0   .0    .0   .0    1423      0    519
```

Figure 6-11 FCX113 UPAGE

Wait State Analysis

Tookit report: FCX114 USTAT

Watch the % CPU and % PGW columns. If values here are not high, guests are not waiting long in a queue for CPU or PAGE. Field descriptions apply to both the USTAT and USTATLOG userid displays. The the exception is the Userid field, which is replaced on UTRANLOG by the field Interval End Time.

User wait states are tested in this sequence:

1. I/O wait
2. Console function wait
3. Instruction simulation wait
4. Page wait

5. CPU wait
6. Running
7. SVM wait and in the eligible list
8. Loading
9. Dormant
10. Dormant and in SVM wait
11. I/O active
12. Test idle wait
13. SVM wait and test idle wait
14. Page active wait
15. Other

6.4.5 Dynamically managing resources in guests

We have already mentioned CP commands that you can use to collect the z/VM and guest information. This section discusses additional commands that you can use.

CP SET QUICKDSP

The Q0 classification is used for critical virtual machines that do not wait for resource availability on the eligible list. Instead, these guests move immediately to the dispatch list. Use **`cp set quickdsp`** to designate a virtual machine as Q0. Or you can add the **`option quickdsp`** statement to the user's directory entry.

Use this command carefully to prevent compromising the system. Imagine the results if several virtual machines in Q0 are dispatched without waiting in the Eligible List. You can use the command freely in VM service machines such as RACF®, RSCS, or TCPIP, or when a guest is facing temporary performance issues.

SHARE settings

Share settings control the priority of guests in the dispatch list when contention for processor time occurs. These settings are a more consistent way to assign processor resources than changing the number of virtual CPUs defined to a guest. When you define a Linux guest user directory entry, define as many virtual CPUs as there are physical CPUs in LPAR. This allows share settings to be used. There are two types of share settings:

- Absolute
- Relative

Absolute share allows a guest access to a specified percentage of CPU resource. With an absolute share of 3%, a guest gets 3% of the CPU, regardless of how many other virtual machines are running. Use this setting for workloads with a well-known usage pattern. Absolute share ensures that a guest is not starved for CPU time, provided that SRM settings allow it to stay in the dispatch queue. The responsiveness of a guest with an absolute share does not slow down as the system load increases.

Relative share defines a guest's importance relative to other running guests. Two virtual machines with a relative share of 100 have equal access to the CPU resources. A guest with a relative share of 200 has twice as much access to CPU resources as a guest with relative share of 100. Use relative share for workloads that do not have a well-defined usage pattern.

See 8.5, "CP set share command" on page 147, for more details.

SRM settings

Virtual machines wait on the eligible list for resources to become available before moving to the dispatch list. If there are no resource constraints, virtual machines move immediately to the dispatch list. SRM settings and virtual machine classification determine resource availability. You can check the guest directory definition to determine how many resources were defined for this guest.

For more information refer to 8.4, “CP scheduler controls” on page 142.

Logical CPUs

If you are using a system with more than one physical processor, assign as many logical CPUs as there are physical CPUs in LPAR. There is no performance benefit in allocating more logical CPUs than available physical CPUs. The logical CPU units of work will simply queue up on the available physical CPUs and be executed in sequence.

However, there is an advantage to defining more logical CPUs than physical CPUs when you are testing multi-threaded code. For example, defining more logical CPUs than physical processors can expose timing and locking problems during a system test.

See Chapter 9, “Tuning processor performance for z/VM Linux guests” on page 157, for more details.

6.4.6 Linux guest virtual machine size

Linux uses all available memory for file system buffers to reduce the likelihood of performing I/O. For this reason, set the virtual machine size for Linux guests to as small as possible. A Linux system that does not require 1 GB of memory will eventually fill any unused memory with file system buffers. In general, more small Linux guests will achieve better response time than fewer large ones.

Main storage is shared among all VM guests, and this can adversely affect the VM scheduler, as we discussed in Chapter 8, “CPU resources, LPARs, and the z/VM scheduler” on page 125.

These guidelines will help you determine the optimal virtual machine size for a Linux guest:

- Initially define the guest virtual machine size to be two-thirds as large as the recommended memory size for the applications running on the guest.
- Define a VDISK or DCSS swap device for the guest. Use a size that is twice the guest virtual machine size.
- Start the guest and run the application with a realistic workload test. Monitor memory consumption and swap device activity during the test.

You can infer optimal virtual machine size from swap device activity:

- If the guest never swaps, the virtual machine size is likely too large and must be reduced.
- If swapping affects performance, the virtual machine size is too small and must be increased.
- If the guest begins to swap as load increases, the guest virtual machine size is correct. Ideally, as the workload reaches steady state, Linux swapping will slow or even stop.

6.5 About DCSS

This section outlines the differences and similarities between various types of segments.

Discontiguous saved segments

Discontiguous Saved Segments (DCSS) is a z/VM technology that saves a portion of guest storage. Many guests can share this storage, which uses the same part of real storage and occupies memory segments (1 MB of real storage) on a MB boundary. The storage is accessed by name and can be embedded above the virtual machine’s defined storage size. DCSS is a powerful tool that provides high levels of resource sharing between guests.

When Linux runs under z/VM, each Linux guest machine loads its own copies of programs and manages its own storage. When multiple copies of Linux are running on the same z/VM system, you can use functions unique to z/VM and System z to share memory pages between the multiple Linux guests. Figure 6-12 on page 87 shows a typical DCSS scheme.

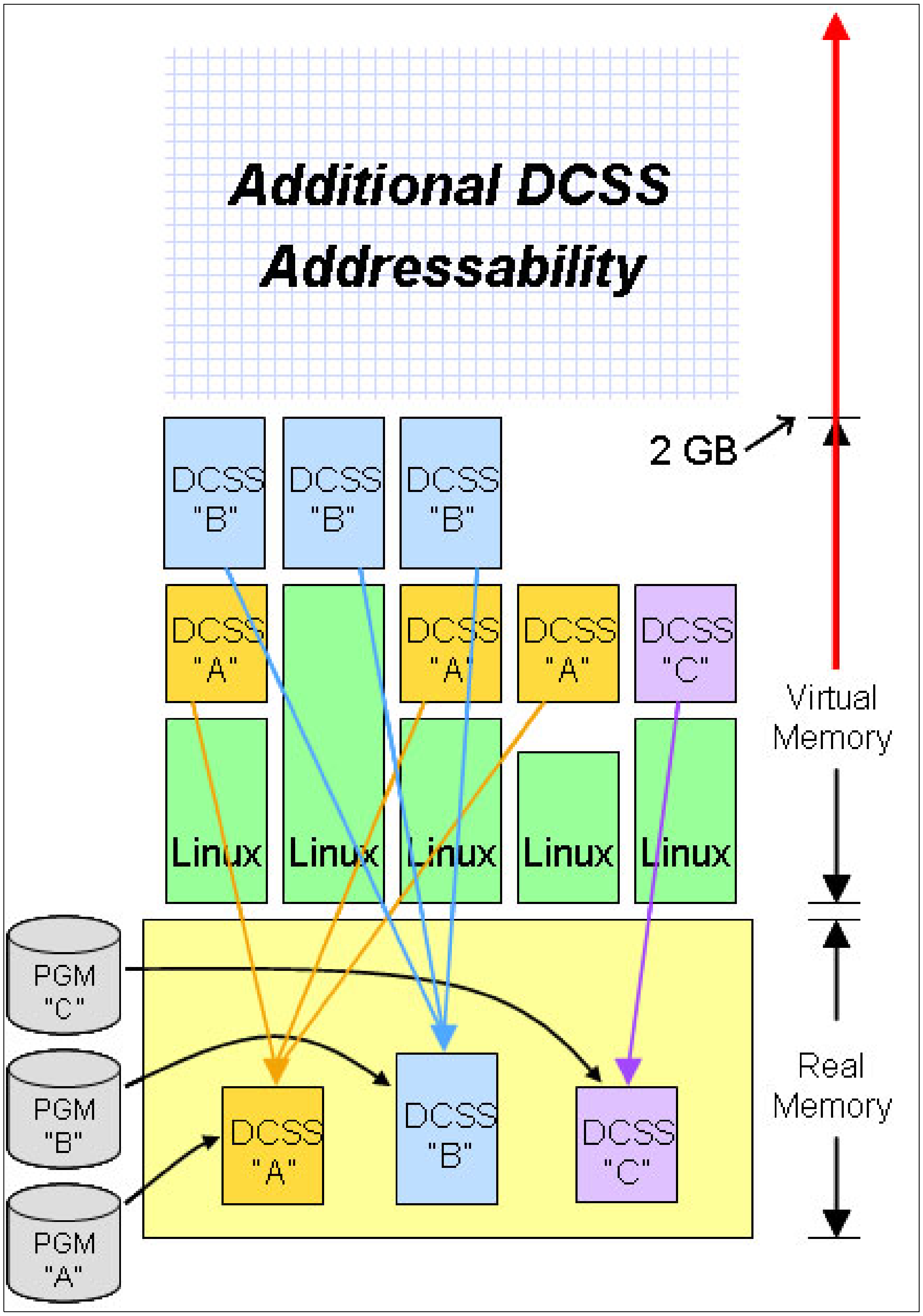


Figure 6-12 DCSS scheme

Review the terms and concepts discussed in the following sections before you begin analyzing your z/VM system.

6.5.1 Saved Segment

Saved Segment is a special z/VM feature of that provides a range of virtual storage pages that you can define to hold data or reentrant code, that is, programs. There are two types of saved segments, SN and SR. SN implements shared real/write access and does not save any data. SR implements shared read-only access.

6.5.2 Named Saved Segment

You can specify a unique name for a saved segment and save it in pageable format on a system data file. Use a named saved segment for the control program portion of an operating system that is IPLed frequently or that is used by many virtual machines. To IPL the NSS, specify the saved system name in the IPL command instead of a device number. The result is that the NSS is paged to the user's virtual storage from a system data file on a CP-owned spooling volume. The NSS is paged on an on demand basis instead of being totally read in from the system residence volume of the operating system during the IPL.

Using NSS eliminates a significant amount of Start Subchannel instruction simulation processing that is normally required to load an operating system, which improves IPL performance. NSS also supports shared storage capability. An example of NSS is the CMS that is used by all CMS users sharing pages in real memory, thus occupying less space than if each virtual machine used a copy.

6.5.3 Segment space

A segment space is a special type of DCSS composed of up to 64 member saved segments referred to by a single name and occupying one or more architected segments. Although individual address ranges are specified on page boundaries anywhere within an architected segment, a segment space begins and ends on a MB boundary. A user with access to a segment space has access to all its members.

6.5.4 Member saved segment

A member saved segment is a special type of DCSS that belongs to a maximum of 64 segment spaces, including ones used for applications. A member saved segment begins and ends on a page boundary and is accessed by its name or by a segment space name. When a virtual machine loads any member of a segment space, the virtual machine has access to all members of the space. You must load all the other members before using any one of them.

The term *discontiguous saved segment (DCSS)* generally refers to saved segments that are neither segment spaces nor member saved segments.

A segment space or member saved segment can contain up to 2047 MB. A DCSS can contain over 2047 MB.

DCSS is a physical saved segment or member saved segment where CMS logical saved segments are defined.

6.5.5 Using saved segments

You can generate a NSS in two steps. First, define a skeleton system data file for the system you will save and name that file using the **`defsys`** command. This command is composed of the following terms:

- Name to be assigned to the NSS
- Minimum virtual storage size for the NSS (MINSIZE operand)
- Pages to be saved and the virtual machine access permitted for each range of pages
- Whether the NSS is restricted
- Range of general registers that will contain the IPL parameter string when the NSS receives control from CP

Your next step is to specify the vmgroup operand, also using **`defsys`**. vmgroup dictates that virtual machines that IPL the NSS become members of a virtual machine group identified as NSS. vmgroup defines the group associated with the NSS.

You can define specific access, as listed below, for each page range in NSS. The *no data saved* term means that the page is treated as a new page when the virtual machine first references it. The contents of this page and its storage key are not saved by the **`savesys`** command:

- Exclusive read/write access
- Exclusive read/write access, no data saved
- Exclusive read-only access
- Shared read/write access
- Shared read/write access, no data saved
- Shared read-only access
- Read/write access by a CP service, shared read-only access by a virtual machine, no data saved

Note: A virtual machine can access a restricted NSS only if you specify **`namesave`** for the NSS in its z/VM directory entry.

6.6 Exploiting the shared kernel

For Linux to scale effectively on z/VM, it needs to share the Linux kernel in NSS (Figure 6-13). The function of sharing the Linux kernel will be released and supported in SUSE service packs.

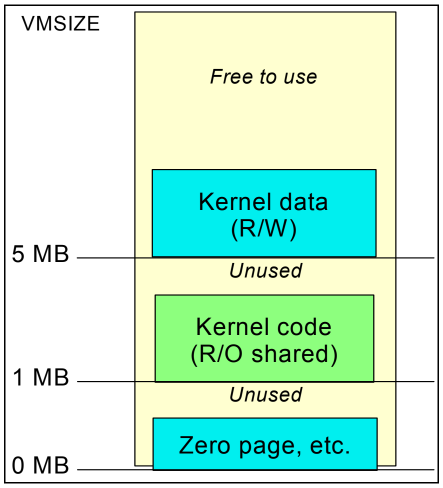


Figure 6-13 Linux memory map with shared kernel

Note: You need to recompile the Linux kernel to create a Linux system that uses NSS support for other SUSE releases or distributions. Be aware that your distributor might not support systems running with a custom compiled kernel, so depending on the level of support required and available, this method might not be appropriate for your installation.

Follow these steps to create a Linux NSS in SLES 10 or 11:

- Define a skeletal system data file (SDF) for the compiled kernel using the **`cp defsys`** command.
- Save the NSS enabled kernel to SDF using the **`cp savesys`** command.

For other releases, follow these steps:

- Compile the Linux kernel and enable the appropriate configuration options for NSS.
- Define a skeletal system data file (SDF) for the compiled kernel using the **`cp defsys`** command.
- Save the NSS-enabled kernel to SDF using the **`cp savesys`** command.

Use the kernel configuration option **`config_shared_kernel`** 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 System Z architecture, protection is assigned on a segment basis. Each segment is 1 MB.

Note: Do not use this option if Linux is running in dedicated memory, for example, on an Intel server or in a System Z LPAR, as that wastes memory dedicated to that server. However, this type of memory cost does not occur if Linux runs on a virtual machine, nor does z/VM need to provide it, as lost memory below 3 MB can be compensated for by increasing the virtual machine memory.

6.6.1 Building an NSS-enabled Linux kernel

Use this process if your current distribution does not support the NSS in the kernel. If you are running a Linux with SLES10 or SLES11, see 6.6.2, “Defining a skeletal system data file for Linux NSS” on page 92.

Follow these steps to build a Linux kernel on System z:

1. Obtain the kernel source code.
2. Obtain and apply the System z specific 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.

For more details about building a Linux kernel for the System Z family, use this link:

http://www.vm.ibm.com/linux/dcss

For NSS support, enable the **`config_shared_kernel`** option (Example 6-1). Use the **`make menuconfig`** command, and then select **Base Setup ∅ VM shared kernel support**.

Example 6-1 Main panel: make menuconfig

```
[*] Enable enterprise support facility
[*] Split the kernel package into multiple RPMs
[ ] Kernel to suit desktop workloads
    General setup  --->
[*] Enable loadable module support  --->
[ ] Infrastructure for tracing and debugging user processes  --->
-*- Enable the block layer  --->
    Base setup  --->
    Power Management  --->
-*- Networking support  --->
    Device Drivers  --->
    File systems  --->
    Kernel hacking  --->
    Security options  --->
-*- Cryptographic API  --->
    Library routines  --->
[*] Virtualization  --->
---
    Load an Alternate Configuration File
    Save an Alternate Configuration File
```

Select the **VM shared kernel support** option in Base setup (Example 6-2).

Example 6-2 Base setup: VM shared kernel support

```
[<M> Kernel support for MISC binaries
[ ] Show crashed user process info
[*] Pseudo page fault support
[*] VM shared kernel support
<M> Cooperative memory management
[*]   /proc interface to cooperative memory management
[*]   IUCV special message interface to cooperative memory management
[*] Linux - VM Monitor Stream, base infrastructure
```

```
<M>   Monitor memory management statistics
<M>   Monitor OS statistics
<M>   Monitor overall network statistics
    Timer frequency (100 HZ)  --->
[*] s390 hypervisor file system support
[*] kexec system call
```

6.6.2 Defining a skeletal system data file for Linux NSS

VIew the generated `system.map` file (Example 6-3) to identify the zero page, shared kernel code, and non-shared kernel data regions in the constructed kernel. In this book, we define a SLES10 NSS, but you can also use SLES11 NSS. Note that the page range for each region is based on the symbols that indicate the start and end of the region.

Note: The address to locate the end of the region is 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.

Example 6-3 Portions of the System.map to determine NSS configuration

```
00000000 A _text
00000298 t iplstart
00000800 T start
00010000 t startup
00010400 T _pstart
00011000 T s390_readinfo_sccb
00012080 T _pend

00100000 T _stext
00100100 t rest_init
00100148 t init

004ef190 r __param_maxchannels
004ef1b8 r __param_format
004ef1e0 R __stop___param
00500000 A _eshared
00500000 D init_mm
00500310 D init_task

005ca000 d per_cpu__flow_flush_tasklets
005ca100 d per_cpu__rt_cache_stat
005ca140 d per_cpu____icmp_socket
005ca148 A __per_cpu_end
005cb000 A __bss_start
005cb000 A __init_end
005cb000 B system_state
005cb008 B saved_command_line
005cb010 B __per_cpu_offset
```

You can identify this information in the `System.map` file (Example 6-3 on page 92).

- The region extending from _text to _pend is the zero page area. Using addresses 0x00000000 and 0x00012080, the page range is 0 - 12.
- The region extending from _stext to the symbol *immediately before* _eshared (__stop___param) is the shared kernel code. Using addresses 0x00100000 and 0x004ef1e0, the page range is 100 - 4ef.
- The region extending from _eshared to _bss_start is the non-shared kernel data. Using addresses 0x00500000 and 0x005cb000, the page range is 500 - 5cb.

Using these values, construct the **`defsys`** command as follows:

```
DEFSYS SLES10 0-12 EW 100-4ef SR 500-5ca EW MINSIZE=24M MACHMODE XA,ESA
```

Table 6-1 displays parameters for this command.

Table 6-1 defsys command parameters

Command	Parameter
SLES10	Identifies the NSS on IPL. By using different NSS names, you have different versions of the kernel saved as NSS on a VM system. IPL identifies the NSS to be used by the virtual machine. Sharing is most efficient, in both memory and system administration, when many virtual machines share the same NSS, so use this command carefully.
0-12 EW	Page range for the zero page region. This region is copied to each virtual machine in exclusive write mode (EW).
100-4EF SR	Page range for the shared kernel code. Each virtual machine uses a shared, read-only copy (SR).
500-5CA EW	Page range for the private kernel data. Each virtual machine uses an exclusive write copy.
MINSIZE=24M	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	NSS can be IPLed in an XA or Enterprise Systems Architecture (ESA) virtual machine.

For more information, use the link below:

http://public.dhe.ibm.com/software/dw/linux390/docu/l26dhe01.pdf

6.6.3 Saving the kernel in Linux NSS

Use the **`savesys`** command to save the NSS-enabled kernel to the skeletal SDF. We will use the **`savelx exec`** script from the How To Use VM Shared Kernel support pages found at the links below:

http://www.vm.ibm.com/linux/linuxnss.html
http://public.dhe.ibm.com/software/dw/linux390/docu/lk39dd11.pdf

We modified the script to use the **defsys** parameters specific to our kernel (Example 6-4). **defsys** original values work best when you create SDF. However, these values create a larger NSS than required and therefore require more IPL time.

Example 6-4 savelx exec to save 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 '0-12 EW 100-4EF SR 500-5CA 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
```

You need the following two parameters for **savelx exec:**

- The name of the Linux NSS. We used SLES10 in our example.
- The DASD device on which the NSS-enabled kernel resides. You must define this DASD for each virtual machine using Linux NSS.

Assuming that the NSS-enabled kernel resides on virtual device 201, we saved Linux using the following command:

```
savelx sles10 201
```

6.6.4 Changing Linux images for NSS shared kernel

Use the **cp ipl** command, with the name of the Linux NSS, to boot Linux NSS in a virtual machine. In our example, the NSS is SLES10, as shown here:

```
IPL SLES10 ipl sles10
```

DASD device numbers for virtual machines using the NSS must match the device numbers used when the NSS was created, even though the NSS kernel is not booted from DASD. 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.

6.7 Execute-in-place technology

You can reduce memory requirements and improve Linux performance with DCSS. In a virtualized environment, Linux images might need the same data at the same moment, which might result in the same data being loaded into memory several times. Linux uses a major chunk of memory for binary application files and for shared library files. Execute-in-place technology uses DCSS managed by z/VM to share memory for applications.

Consider the following when you plan to share data:

- Application files can only be shared by Linux instances that use the same version of an application.
- Shared data must be read-only.
- Applications and libraries that are frequently used by numerous Linux instances are good candidates.

You can define DCSS above Linux guest storage or in a storage gap. Usually, DCSS in a storage gap is the preferred placement for a 64-bit Linux guest. Linux can access a DCSS through the execute-in-place file system (xip2).

Linux loads shared library and application files through the mmap() operation, which maps the file contents to the application address space. The xipl file system performs this operation by mapping the DCSS contents to the application address space, while other file systems use a page cache. This feature is called execute-in-place, because the file contents are not physically copied to a different memory location. Using execute-in-place technology, Linux can access files and libraries without I/O operations, which increases overall performance. Linux can also run applications directly in a shared memory segment, which saves memory.

6.8 Setting up DCSS

We will walk through the process of setting up a DCSS, including planning, creating, defining, and testing. Your first step is to set up the DCSS. Table 6-2 outlines DCSS locations and maximum sizes.

We use the `/us` directory and the **`du -sk`** command to show directory size. In our example, the directory is 1035264 KB, but we increased it to 1258291 KB (1.2 GB) to manage metadata and future space for software updates. This makes the DCSS size above the guest size.

Table 6-2 Maximum DCSS size

Kernel	DCSS location	Maximum DCSS size
64-bit	Storage gap	1919 MB
64-bit	Above guest storage	2047 MB
31-bit	Either location	1960 MB

You can now determine the start and end addresses of the DCSS. These addresses must be on a page boundary (4 KB) and in hexadecimal format. We used these values in our example:

- Start address: 512 MB = 0x20000000
- End address: 512 MB + 1011 MB - 1 MB = 0x5F2FFFFF
- Start frame: 0x20000
- End frame: 0x5F2FF

Before the Linux kernel can use the DCSS above the guest storage, it must be aware of the extended address space that covers the DCSS. To do that, add a mem=<value> on your kernel parameter file, run zipl, and reboot it. In our example, we used this value:

```
mem=1523M
```

6.8.1 Creating the over mounting script

This step illustrates the usage of an entire directory. This requires that you overmount the DCSS on the directory that is included in the segment. This directory resides on the root partition, so you need a script to run before system startup.

IBM provides a sample script, xipinit.sh, that you can customize to your requirements. Example 6-5 shows the xipinit.sh script tailored for our example. The parameters we customized are shown in **bold** and described below.

- ROMOUNT

  The directory where you need to mount DCSS. For the first run, we used `/mnt`.

- XIPIMAGE

  The name of DCSS. We used USRXIP.

- RODIRS

  The directory included on the DCSS.

Example 6-5 xipinit.sh

```
#!/bin/sh
#
# xipinit.sh, Version 2
#
# (C) Copyright IBM Corp. 2002,2005
#
# /sbin/xipinit: use read only files from another file system
#
# default options
# internals
#set -v
#FAKE=echo
#mount point of xipimage
ROMOUNT="/mnt"
#name of xipimage
XIPIMAGE="USRXIP"
#name of device node
RODEV="/dev/dsccblk0"
RODIRS="/usr,"
# make sure it ends with ,
RODIRS="$RODIRS",
mount -t sysfs none /sys
if [ ! -e /sys/devices/dcssblk ]; then
echo "xipinit: loading dcssblk module"
/sbin/modprobe dcssblk
fi
echo $XIPIMAGE > /sys/devices/dcssblk/add
# mount ro file system to its mount point
echo "xipinit: mounting read-only segment"
$FAKE mount -t ext2 -o ro,xip "$RODEV" "$ROMOUNT"
```

```
# bind mount all ro dirs into rw file system
while [ -n "$RODIRS" ] ; do
dir="${RODIRS%%,*}"
RODIRS="${RODIRS#*,}"
test -d "$dir" || continue
echo "xipinit: binding directory" $dir
$FAKE mount --bind "$ROMOUNT/$dir" "$dir"
done
umount /sys
# run real init
$FAKE exec /sbin/init "$@"
```

6.8.2 Creating a DCSS

Use the block device driver to create the segment. First make sure that the Linux instance has class privilege E. Follow these to build the DCSS:

1. Shut down your Linux image and IPL the CMS.
2. Define the DCSS with this command:

   ```
   defseg usrxip 20000-5F2FF sr
   ```

3. Define the storage guest to cover the entire DCSS. Two GB will cover any possible DCSS with this command:

   ```
   define stor stor 2048M
   ```

4. Save the DCSS with this command (can take a long time to run):

   ```
   saveseg ysrxip
   ```

6.8.3 Copying data to DCSS

Use the block device driver to copy data to DCSS. This step is necessary because data from `/usr` must be copied inside the new file system. You cannot use the `xip2` file system for this step because it is a read-only file system. In our example, we used ext2, which is similar and compatible in terms of structure. Use this in read/write mode, then unmount and remount with the xip options.

Follow these steps to copy data to DCSS:

1. Load the DCSS module with this command:

   ```
   # modprobe dcssblk
   ```

2. Create the node for the `/dev` directory structure. The major number is indicated in `/proc/devices` with the command:

   ```
   # mknod /dev/dcssblk0 b 252 0
   ```

3. Load the DCSS with this command:

   ```
   # echo "usrxip" > /sys/devices/dcssblk/add
   ```

4. Change the access mode from share to exclusive-writable with this command:

   ```
   # echo 0 > /sys/devices/dcssblk/usrxip/shared
   ```

5. Create an ext2 file system with this command:

   ```
   # mke2fs -b 4096 /dev/dcssblk0
   ```

6. Mount the file system on /mn with this command:

   ```
   # mount /dev/dcssblk0 /mnt/
   ```

7. Copy the contents of /usr to /mnt with this command:

   ```
   # cp -a /usr/* /mnt
   ```

8. Save the DCSS using this command:

   ```
   # echo 1 > /sys/devices/dcssblk/usrxip/save
   ```

   You might see the z/VM console message `Segment USRXIP is currently busy, it will be saved when it becomes idle`. The segment is saved when you umount the file system.

9. Umount the /mnt file system using this command:

   ```
   # umount /mnt
   ```

6.8.4 Testing DCSS

You now need to test the DCSS that you created by mounting the xip2 file system. Use this command:

```
# mount -t ext2 -o ro,xip /dev/dcssblk0 /mnt
```

Make sure that the file system is mounted correctly by checking the /proc/mounts file systems (Example 6-6).

Example 6-6 mount point

```
lnxsu4:/ # cat /proc/mounts
rootfs / rootfs rw 0 0
udev /dev tmpfs rw 0 0
/dev/dasdb1 / ext3 rw,data=ordered 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
debugfs /sys/kernel/debug debugfs rw 0 0
devpts /dev/pts devpts rw 0 0
/dev/dcssblk0 /mnt ext2 ro,nogrpid,xip 0 0
```

6.8.5 Activating execute-in-place

In a large Linux server farm on z/VM, execute-in-place technology increases overall performance. Before you continue, verify that the scripts shown in Example 6-5 on page 96 work correctly, following the steps below. The script is located in the saved /sbin directory and shown in Example 6-7.

1. At the end of the script, use the command **cat /proc/mount** to confirm that the directory is mounted.
2. Umount the file system and add the following line in the parameter line in /etc/zipl.conf:

   ```
   init=/sbin/xipinit.sh
   ```

3. Save zipl and reboot your Linux system.

Example 6-7 xipinit.sh script test

```
# /sbin/xipinit.sh
# cat /proc/mounts
udev /dev tmpfs rw 0 0
/dev/dasdb1 / ext3 rw,data=ordered 0 0
```

```
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
debugfs /sys/kernel/debug debugfs rw 0 0
devpts /dev/pts devpts rw 0 0
```

If the file system is mounted, you can mount the entire `/usr` file system in DCSS by changing the script from the `/mnt` directory to `/usr`.

For more details about using DCSS, use these links:

http://www.vm.ibm.com/linux/linuxnss.html
http://public.dhe.ibm.com/software/dw/linux390/docu/lk39dd11.pdf

6.9 Cooperative Memory Management (CMM)

CMM allows an external entity, for example, the z/VM resource monitor (VMRM) or a third-party external monitor, to reduce the storage size of a Linux guest. You must load the Linux Kernel module CMM to allow collaboration with the external monitor. The external monitors track how the Linux guests cope with their memory requests. Linux guests receive information about how much free memory is available through the interfaces created with CMM (Figure 6-14). If not enough free memory is available, the Linux guest starts to clean the page cache, the slab cache, or other memory until it meets the threshold requirement. Under pressure, the Linux guest might start swapping. CMM is sometimes referred to as CMM1.

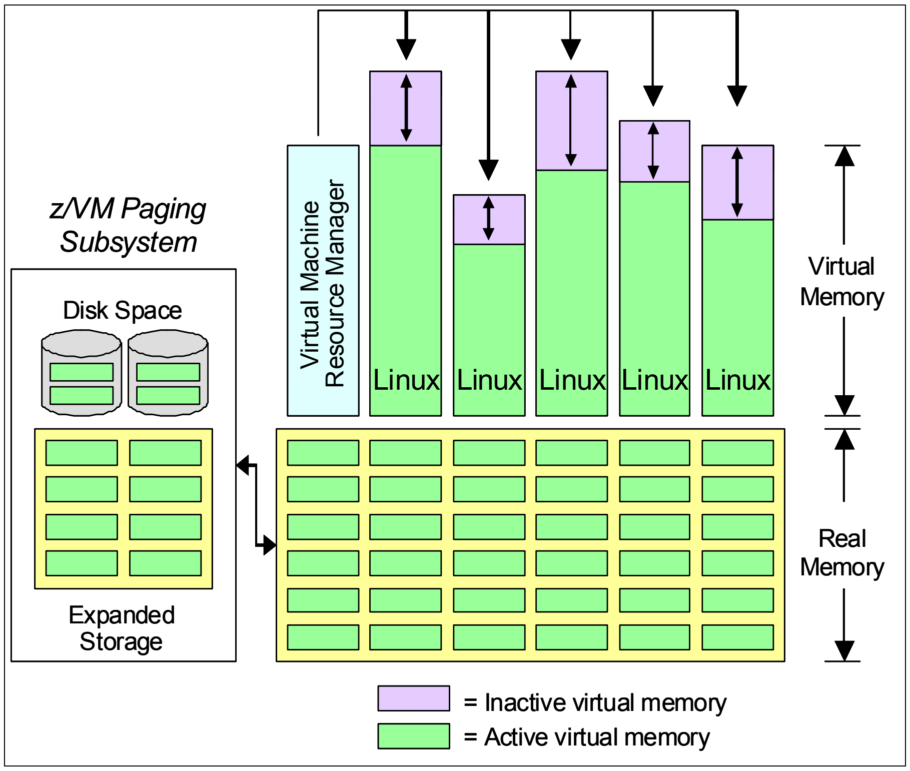


Figure 6-14 VMRM Cooperative Memory Management (VMRM-CMM)

Setting up CMM

You need to load the module *cmm* in the involved Linux guests. Make sure that VMRMSVM has been configured and started in z/VM. Then complete these steps:

1. Create a VM Resource Manager (VMRM) configuration file (named, for example, VMRM CONFIG A1) on the VMRMSVM user ID to set up z/VM to use CMM. In our example, the file contained these statements (the asterisk is a wild card):

   ```
   NOTIFY MEMORY LNX001 LNX002 LNX003 LNX004 LNX005
   Or:
   NOTIFY MEMORY LNX00*
   ```

2. This statement sends VMRMSVM to the MAINT user ID:

   ```
   ADMIN MSGUSER MAINT
   ```

   Add it to receive VMRMSVM messages.

3. After you create the configuration file on the VMRMSVM A-disk, log off VMRMSVM. Use xautolog VMRMSVM from the user MAINT to start the VMRM server.
4. To stop the VMRMSVM server, log on to VMRMSVM and issue the **hmonitor** command (or FORCE VMRMSVM as user MAINT).
5. Use the **modprobe** command to load the CMM kernel extension on the Linux servers:

   ```
   modprobe cmm
   ```

For more details, see *IBM Linux on System z - Device Drivers, Features, and Commands*, SC33-8411:

http://download.boulder.ibm.com/ibmdl/pub/software/dw/linux390/docu/lk39dd11.pdf

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

6.10 Collaborative memory management assist (CMMA)

Linux guests and the z/VM control program (CP) exchange information regarding memory usage. This exchange allows both the guests and the z/VM host to optimize their memory management in the following ways:

- CP knows when a Linux application releases storage, and can then select those pages for removal at a higher priority, or reclaim the page frames without the overhead of paging-out their data content to expanded storage or disk.
- CP recognizes clean disk cache pages. Linux can reconstruct this page content, which allows CP to bypass paging-out the data contents when reclaiming the backing frames for these pages. If Linux or its application subsequently tries to refer to the discarded page, Linux is notified that the page has been discarded and can reread the contents from disk or otherwise reconstruct them.

- Host Page-Management Assist (HPMA), with CMMA, allows the machine to supply fresh backing page frames for guest memory when the guest reuses a previously discarded page (Figure 6-15). This eliminates the need for the z/VM Hypervisor to intercept and resolve these host page faults.

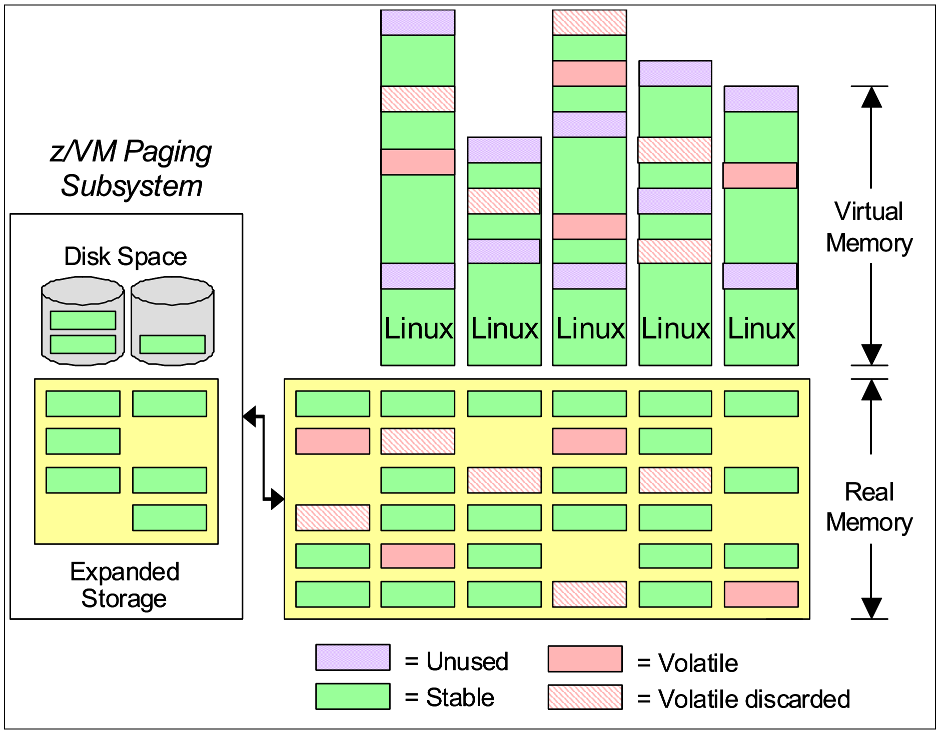


Figure 6-15 Collaborative memory management assist

Pages are flagged differently in Linux system using CMMA, as compared to those not using CMMA. Beginning with the boot process, when the page cache gets filled, each page gets flagged as stable or as volatile. Volatile flagged pages can get stolen by z/VM and provided to other guests. This is not apply to stable pages.

Read more details at *IBM Linux on System z - Device Drivers, Features, and Commands*, SC33-8289, or use this link:

http://www-01.ibm.com/common/ssi/ShowDoc.jsp?docURL=/common/ssi/rep_ca/9/877/ENUSZP09-0459/index.html&lang=en

Setting up CMMA

You need z/VM 5.3 or later and System z family servers to use CMMA. IPL Linux with the option cmma=on. This causes the kernel to use special page flags. By default, CMMA is off.

Read more details in *IBM Linux on System z - Device Drivers, Features, and Commands*, SC33-8411:

http://download.boulder.ibm.com/ibmdl/pub/software/dw/linux390/docu/lk39dd11.pdf

7

Linux swapping

In this chapter, we discuss Linux swap files, including:

- Swapping basis
- Swap device options
- Peak situations swapping versus regular swapping
- Swapping recommendations

7.1 Linux swapping basics

Maintaining the correct amount of memory for a Linux system is a complicated task. Linux will use allocated memory to run processes, and use any remaining memory to cache data. So over time, Linux will use all available memory. One solution to this problem is to allot the virtual machine only the memory amount that it needs. This is tricky because memory requirements vary over time as processes start and stop. The result is a virtual machine that is either too large or too small.

Linux begins to swap when the virtual machine is too small for the workload. Pages from some processes are moved out and stored on Linux swap disks to make room for process that need to run. Occasional swapping is not detrimental, but continuous swapping affects performance. The acceptable amount of swapping depends on the efficiency of the swapping mechanism.

We used the hogmem program in a virtual machine to compare the efficiency of Linux swap devices. The hogmem program allocates virtual memory for a process and then sequentially accesses each page. Linux begins to swap pages when the amount of memory allocated by hogmem exceeds the free memory in Linux. The hogmem program text is shown in 7.9, “hogmem program text” on page 122.

Linux began continuos swapping when we ran enough hogmem programs in parallel and allocated more virtual memory than what Linux could free up to use. Keep in mind that constant swapping slows performance.

We learned that nearly half of the 128 MB must be available for processes by running an experiment where we IPL the kernel in a 128 MB virtual machine, using the **free -m** command (Example 7-1). The command output shows that 60 MB is not in use. Linux will reclaim much of the buffers and cache when necessary, so 87 MB (free + buffers + cached) is the amount that we can obtain without causing much swapping.

Example 7-1 Available memory in an idle 128 MB virtual machine

```
# free -m
             total       used       free     shared    buffers     cached
Mem:           116         55         60          0          3         24
-/+ buffers/cache:         27         89
Swap:         7043          0       7043
```

We repeated the experiment using 87 MB process and received the memory usage shown in Example 7-2. The results indicate that Linux did not give up all buffers and cache, but began swapping instead.

Example 7-2 Memory usage in 128 MB machine with 87 MB process

```
# free -m
             total       used       free     shared    buffers     cached
Mem:           116        115          1          0          0          4
-/+ buffers/cache:        110          5
Swap:         7043          6       7036
```

Linux obtained 59 MB from the free pool, leaving 1 MB available to satisfy sudden and urgent memory requests. The remaining 28 MB was obtained from buffers/cache and by memory freed by swapping.

Using the **vmstat** commands showed little swapping while the process runs (Example 7-3).

Example 7-3 Monitoring swap activity

```
# vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-------cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0  56392   4344  28680    0    0    49    13   13   24  1  2 97  0 0
 0  0      0  56364   4344  28680    0    0     0     8   16   10  0  0 100  0 0
 0  0      0  56364   4344  28680    0    0     0     0    6    3  0  0 100  0 0
 0  0      0  56364   4344  28680    0    0     0     0    6    5  0  0 100  0 0
 0  0      0  56364   4344  28680    0    0     0     0    6    7  0  0 100  0 0
 1  0      0  12724   4352  28672    0    0    16    44   35   28 19  3 78   0 0
 1  1  40884   1476    116   1560    0 62316  2580 62364 1868 383 16 18  0  66 0
 0  1  62316   4324    128   2764  192    0  1768     0  103   37  1  2  0  97 0
 0  3  62316   1560    128   2960 3736    0  3736     0  156  301  2  1  0  97 0
 0  3  62316   1680    128   3056 1372    0  1372     0  176  310 25  2  0  73 0
 0  1  81708  11240    104    664 3092 34532  3652 34580 2339 850  1  5  0  93 0
 1  1  81708   5808    104   1120 5176    0  5608     0  703  730 32  5  0  62 1
 1  0  81708   5500    104   1304  332    0   332     0  169  135 97  0  0   3 0
 1  0  81708   5140    108   1720   96    0   464     0  113   24 100 0  0   0 0
 1  0  81708   5140    108   1720    0    0     0     0  105    9 99  1  0   0 0
 1  0  81708   5140    108   1720    0    0     0     0  105   13 100 0  0   0 0
 1  0  81708   5140    108   1720    0    0     0     0  107    9 98  1  0   0 1
```

7.2 Linux swap cache

Linux tries to reduce the I/O burden even during the swap cache implementation by occasional swapping. The copy of the swapped-out page is held on the swap device or file after the page is eventually switched. There are two options if the page is swapped out for the second time. If the page was is not dirty (not changed), it can be discarded. If the page is dirty (already changed), it is brought out a second time. The page already has a location in the swap space assigned that reduces the overhead for assigning a new location.

Note: The Linux swap cache sometimes causes misleading swap-in and swap-out page information in monitoring commands, for example, when you use tools such as **vmstat**. The following paragraphs often show more pages swapped in than swapped out.

7.3 Linux swap options

Linux on System z has many options for choosing the a swapping device. Often installations use BVDISK when running on z/VM, and ECKDDASD when running in LPAR without z/VM. There are new options like the DIAGNOSE support for 64-bit DASD, supported on z/VM 5.2 or later, that were available only in 31-bit. We recommend that you choose a swap device based on whether swapping will occur regularly during everyday processing, or whether swapping will occur only occasionally. Keep in mind that, when swapping begins, performance is dependent on how quickly pages can be transferred to and from the swap device.

We measured the swap speed of five options (Table 7-1). We used three disk types and four drivers, which resulted in seven valid combinations.

Table 7-1 Swap speed of five options

Driver/device type	DASD	VDISK	SCSI disk
ECKD™	Default	N/A	N/A
DIAGNOSE[a]	Alternative	Alternative	N/A
FBA	Alternative	Default - ldl format	Alternative
FCP	N/A	N/A	Default

a. DIAGNOSE is only available if Linux is running as a z/VM guest.

7.4 Swapping to DASD

We set the baseline for our benchmark by measuring the performance of the extended count key data (ECKD). We needed to increase memory to cause more swapping. If increased enough, we can obtain a constant swap rate that we can then study with measurement tools. In this test, we used 3390-9 DASDs on an ESS-800.

7.4.1 Swapping to DASD with ECKD

Increasing the virtual memory to 128 MB results in the output shown in Example 7-4. Because hogmem walks through the allocated memory sequentially, we forced all pages from swap into memory and out again to swap.

Note that the swap rate is reported in columns si (that is, memory swapped in from disk) and so (that is, memory swapped out to disk). Both values report the number of 1 KB blocks transferred. Use the **`vmstat 1`** command to catch values every second and show the non-steady swapping. The **`vmstat 60`** command generates average value per minute. Use this command when you need swapping numbers as more constant over time.

Example 7-4 Swapping during memory usage test

```
# vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0      0   2208   3084  21168    0    0  1340    37  117  206  5  7 61 28  0
 0  0      0   2208   3084  21168    0    0     0     0   10    4  0  0 100  0  0
 0  0      0   2208   3084  21168    0    0     0     0    9    7  0  0 100  0  0
 0  0      0   2208   3084  21168    0    0     0     0    4    3  0  0 100  0  0
 0  5   4544   1292    272   2784    0 13544  3012 13544  145  324  8  7 30 54  1
 0  4  49224   1108    104   1848  416 35700  9028 35800  801  753 15 16  0 69  0
 1  1  80940   1808    104    940  412 31780  4428 31780  209  492 14 14  0 71  1
 0  3 111156   1060    108   2528 4532 30288 14720 30288 1515  781 13 15  0 72  0
 0  1 136776  11432    104    348 3332 29648  5544 29648  295  410  1  5  0 93  1
 0  2 137436   1052    108   1736 20820  796 22876   796  809 1424  8  6  0 86  0
 0  1 138368   2968    108    192 10008 30364 13600 30364 2116  812  4 11  0 86  0
 0  2 138368   1224    104   1808 18776   44 21700    44  864 1317  7  6  0 88  0
 0  2 138368   1648    104    444 13564 31916 15052 31916 1328 1013  4 11  0 84  1
 0  3 138368   1064    104   2716  1384    0  3004     0  134  148  1  2  0 97  0
 0  2 138368   1224     104     256 24204     0 24720      0 1976 1604  8  6  0 86  0
 0  1 138368   6268     108     668 8112 31748  9748 31748  456  662  3 10  0 88  0
```

In Example 7-5, we used the z/VM Performance Toolkit to retrieve swap results. Unlike the results in Example 7-4 on page 106, the numbers here are aggregated over a period of one minute. We measured a swap rate (sum of swapped-in and swapped-out pages) of approximately 4 MBps for the ECKD DASD combination. If we compare the swapped-in and swapped-out values, we see that the first minute has a higher swap-out than the swap-in. The pages are written to disk, and not all of them have to be brought back. In the fifth minute, we see fewer pages written to the swap devices. This is the effect of the swap cache. Some of the pages are already on the swap device, so there is no need to transfer them provided that they are not dirty pages.

Example 7-5 Memory at swapping using ECKD DASD

```
<-Memory(MB)->  <------- Swapping -----> <-BlockIO->
Interval   <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
  Time    M_Total %MUsed   S_Total %SUsed  In  Out   Read  Write
16:41:05  53.8     95.9    7043       0    0    0  0.138  6.138
16:42:10  53.8     98.1    7043     1.9 1818 2205   8825   8823
16:43:11  53.8     97.7    7043     1.9 2488 2303  12018   9224
16:44:10  53.8     98.2    7043     1.9 2715 2586  12818  10346
16:45:10  53.8     97.9    7043     1.9 2401 2153  11196   8620
16:46:11  53.8     98.1    7043     1.9 2087 1992   9671   7968
16:47:11  53.8     95.6    7043     1.9 2059 1951   9727   7807
16:48:10  53.8     97.3    7043     1.9 2012 1870   9404   7483
16:49:10  53.8     97.0    7043     1.9 2021 1844   9371   7379
16:50:10  53.8     97.9    7043     1.9 2057 1959   9636   7839
16:51:11  53.8     97.9    7043     1.9 2184 2077  10244   8310
```

In Example 7-6, we see that the CPU is waiting for I/O to finish more than 90% of the time. Therefore, we conclude that the throughput of our test application is down to 5% or less due to swapping.

Example 7-6 CPU utilization at swapping using ECKD DASD

```
Interval   <-------CPU Utilization (%)----------->
Time       TotCPU User  Kernel IOWait Idle  Stolen
16:41:05    0.4   0.3   0.1      0.1  99.5      0
16:42:10    8.7   3.5   4.2     69.3  21.9    0.2
16:43:11   10.4   4.2   4.9     89.4     0    0.2
16:44:10   10.8   4.2   5.2     89.0     0    0.2
16:45:10    9.3   3.7   4.5     90.5     0    0.2
16:46:11    8.1   3.1   4.0     91.7     0    0.1
16:47:11    8.1   3.1   4.1     91.8     0    0.2
16:48:10    7.6   3.0   3.7     92.3     0    0.2
16:49:10    7.6   3.0   3.6     92.3     0    0.1
16:50:10    7.9   3.0   3.9     91.9     0    0.2
16:51:11    8.5   3.2   4.3     91.3     0    0.2
```

We used an extreme and artificial benchmark that touched many pages. A real application should not be as affected by swapping because it spends more time working with available pages, and not much time is reported waiting for involuntary wait.

Effect of multiple processes swapping to ECKD DASD

Many processes normally run on Linux production systems, which means that more than one process uses swapping when not enough memory is available. In Example 7-7, we ran three hogmem programs in parallel, each allocating 50 MB, and then measured the ECKD DASD swap option. The swap disk is one 3390-9 DASD on an ESS-800 (). Figure 7-1 shows the effect is further.

Example 7-7 Memory when three processes swap using one ECKD DASD

```
<-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval   <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time     M_Total %MUsed  S_Total %SUsed  In  Out  Read  Write
18:23:10  53.8    97.3    7043    2.2  2452 2849 11386  11398
18:24:10  53.8    97.7    7043    2.2  3298 2996 15282  11986
18:25:10  53.8    94.6    7043    2.2  3205 3098 14627  12393
18:26:10  53.8    98.1    7043    2.2  2863 2727 12146  10909
18:27:11  53.8    97.8    7043    2.2  3024 2883 12830  11532
18:28:10  53.8    97.8    7043    2.2  2838 2663 12065  10652
18:29:11  53.8    98.1    7043    2.2  3094 2848 13330  11392
18:30:10  53.8    98.1    7043    2.2  3181 2848 13965  11398
18:31:10  53.8    96.9    7043    2.2  3085 2937 13096  11749
18:32:10  53.8    98.2    7043    2.2  2968 2742 12453  10969
18:33:10  53.8    98.0    7043    2.2  3028 2779 12755  11114
```

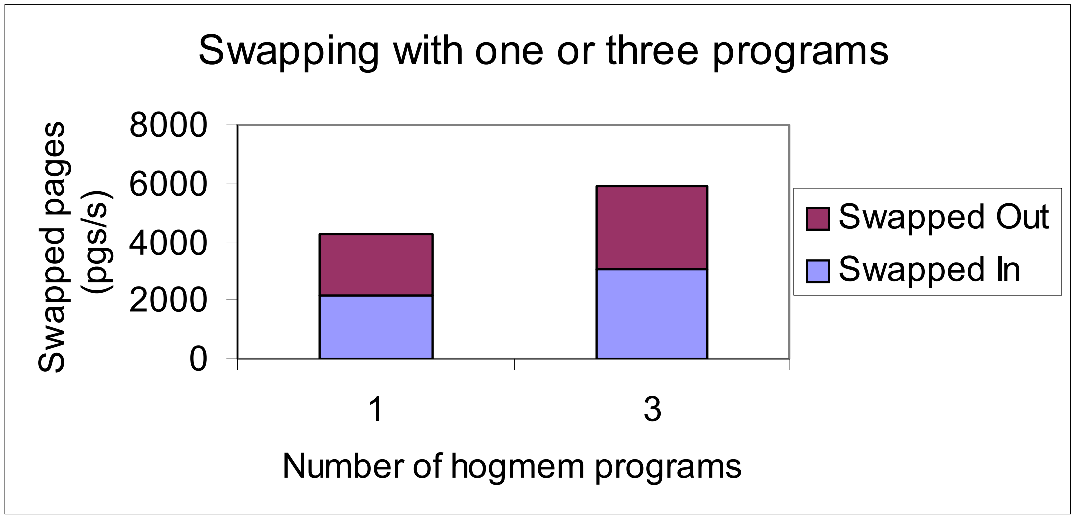


Figure 7-1 Swap rate when one or three hogmen programs swap using one ECKD DASD

Note that this test shows a higher swapping rate than the single process shown in Example 7-5 on page 107. This indicates two things:

- The CPU is not waiting as long for I/O to complete.
- The overall throughput of running three hogmem programs is higher than that of the single hogmem.

We can conclude that the Linux swapping mechanism was able to deal with multiple user requests simultaneously.

Using multiple DASD swap devices

You can use more than one swap device at a time. By default, swap devices are set online (swapon) with different priorities and are then used one after the other. We repeated our test again, running one hogmem program but using two identical ECKD DASD devices of the same swap priority (Example 7-8). It is important to use ECKD DASD from different ranks of the storage server (Figure 7-2).

Example 7-8 Memory when one process swaps using two ECKD DASDs

```
           <-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval    <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time     M_Total %MUsed  S_Total %SUsed  In  Out  Read  Write
18:45:11   53.8   97.6    4696    2.9   938 1152  4628  4620
18:46:10   53.8   98.1    4696    2.9  2367 2154  10921 8616
18:47:10   53.8   97.2    4696    2.9  2449 2196  11188 8783
18:48:10   53.8   97.8    4696    2.9  2503 2176  11682 8705
18:49:10   53.8   97.8    4696    2.9  2533 2345  11536 9380
18:50:11   53.8   98.1    4696    2.9  2451 2150  11152 8598
18:51:11   53.8   97.1    4696    2.9  2336 2085  10812 8340
18:52:11   53.8   97.8    4696    2.9  2519 2237  11403 8947
18:53:11   53.8   98.0    4696    2.9  2402 2128  11045 8514
18:54:11   53.8   98.0    4696    2.9  2329 2143  10768 8571
```

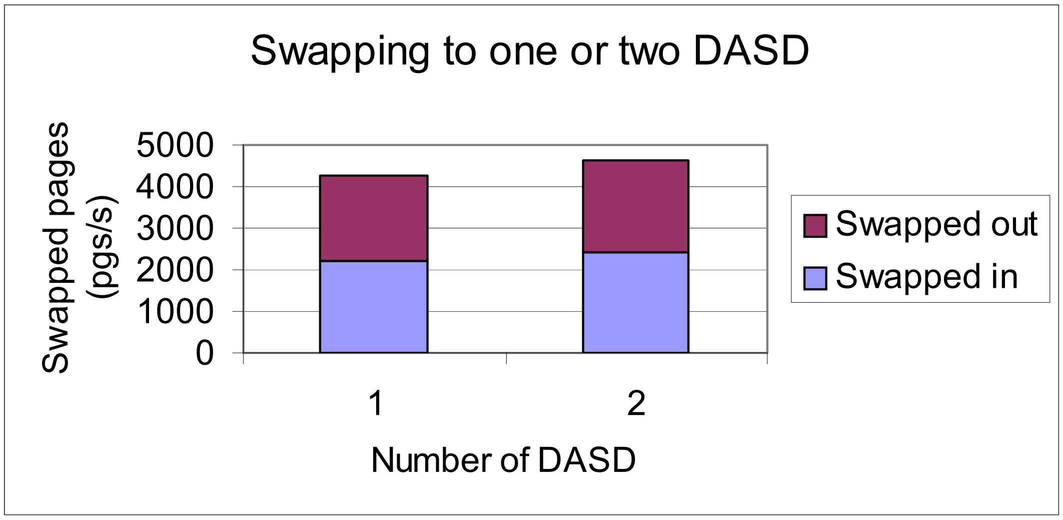


Figure 7-2 Swap rate with one hogmem program and two ECKD DASD

We see a higher swapping rate in this test than in the single hogmem program in Example 7-5 on page 107. The overall throughput of the hogmem program is higher when writing to two disks with the same swap priority. Linux swapping can deal with several swapping devices simultaneously.

We were not able to test two ECKD DASDs with multiple hogmen programs due to time constraints. We assume, however, that more hogmem programs running parallel would need multiple ECKD DASD swap devices to achieve the most benefit. We used ECKD DASD in this test because it is widely used in the field, but would expect similar results with other physical devices such as FBA DISK or SCSI disk attached over FCP.

7.4.2 Swapping to DASD with DIAGNOSE

The Linux DASD driver uses ECKD and FBA disciplines. The DIAGNOSE discipline had problems and was not widely used. Thus, SUSE and Red Hat did not include DIAGNOS in their kernels. However, this condition has improved, and recent distributions include the required dasd_diag_mod module and are able to access DASD using DIAGNOSE. The latest distributions offer the DIAGNOSE option for 64-bit, but do require z/VM 5.2 or later.

In Linux drivers with the DIAGNOSE access method, the CP performs I/O operations when necessary using uses a high-level block I/O protocol (DIAG 250). You will realize more benefits with this protocol than if you allow the driver to choose defaults.

Enabling DIAGNOSE I/O for the 3390 DASD

Follow these steps to enable DIAGNOSE I/O for 3390 DASD:

1. Make sure that the disk is initially formatted under ECKD in the DASD driver. This disk was formatted with CMS and reserved (noted by the keyword RESERVE), but you can also use the Linux commands l**`dasdfmt`** and **`mkswap`**.
2. Change to the `/sys/bus/ccw/devices/<addr>` directory.
3. Use **`echo 0 > online`** to instruct DASD to stop using the device.
4. Ensure that the DIAGNOSE discipline module (dasd_diag_mod.o) is loaded.
5. Use **`echo 1 > use_diag`** to enable DIAGNOSE.
6. Use **`echo 1 > online`** to instruct DASD to use the device again.

DASD will now be used by the DIAGNOSE access method.

Swapping with DIAGNOSE

In this experiment, we used one hogmem program and allocated 128 MB in a 64 MB z/VM Linux guest. See Example 7-9 for the output. We see similar values for swapped-in and swapped-out pages compared to the ECKD DASD measurement seen in Example 7-5 on page 107.

Example 7-9 Memory when one hogmen program is swapping using DIAGNOSE method to DASD

```
           <-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval   <--- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time     M_Total %MUsed  S_Total %SUsed   In  Out  Read Write
17:09:10  53.8    93.8    2348    5.8   2074 2528 10234 10117
17:10:10  53.8    98.1    2348    5.8   2130 1955 10402  7823
17:11:10  53.8    98.1    2348    5.8   2379 2172 11235  8690
17:12:11  53.8    97.9    2348    5.8   2498 2468 12190  9876
17:13:10  53.8    97.4    2348    5.8   2387 2187 11376  8752
17:14:11  53.8    97.8    2348    5.8   2146 2114 10222  8458
17:15:10  53.8    97.6    2348    5.9   2335 2104 11168  8436
17:16:11  53.8    97.7    2348    5.8   2465 2298 14108  9592
17:17:10  53.8    97.8    2348    5.8   2150 2084 10379  8339
17:18:11  53.8    97.5    2348    5.8   2132 2029 10325  8119
17:19:10  53.8    98.6    2348    5.8   2225 2091 10453  8365
```

In Example 7-10, we see a decrease in time spent waiting for I/O and an increase in user time. This indicates that the throughput of a user program using DIAGNOSE to DASD has also increased and that not much time is reported waiting for involuntary wait (that is, stolen).

We ran a service in Linux between 17:15 and 17:16, creating a peak, which was not related to the swap measurement. It was service using a slow priority, or low nice value set, which did not degrade swapping.

Example 7-10 CPU with one hogmen program swapping to DASD using DIAGNOSE

```
Interval  <-------CPU Utilization (%)----------->
Time      TotCPU User  Kernel IOWait Idle  Stolen
17:08:09    0.3  0.1     0.1      0  99.7      0
17:09:10    9.7  4.1     4.7   82.4   7.7    0.2
17:10:10    8.2  3.2     4.0   91.7     0    0.2
17:11:10    9.1  3.7     4.4   90.7     0    0.2
17:12:11    9.9  3.9     4.9   89.9     0    0.2
17:13:10    9.1  3.7     4.4   90.7     0    0.2
17:14:11    8.4  3.4     4.1   91.4     0    0.1
17:15:10    9.6  3.7     4.7   90.2     0    0.2
17:16:11   25.5  3.7    10.1   73.8     0    0.7
17:17:10    8.6  3.4     4.2   91.3     0    0.2
17:18:11    8.5  3.3     4.2   91.3     0    0.1
17:19:10    8.6  3.4     4.2   91.2     0    0.2
17:20:10    9.1  3.5     4.5   90.7     0    0.2
```

7.4.3 Swapping to VDISK

z/VM VDISK is widely used as the swap device for Linux virtual machines. A VDISK is presented to the virtual machine as an emulated fixed block architecture (FBA) DASD device, a virtual device of type 9336. Actually, 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 primary address space and the VDISK address space.

7.4.4 Swapping to VDISK with FBA

For this test, we used a VDISK accessed by FBA, allocating 128 MB in a 64 MB z/VM Linux guest, and with one hogmem program to drive the test. Example 7-11 shows the test results.

Example 7-11 Memory swapping using FBA VDISK

```
            <-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval     <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time        M_Total %MUsed  S_Total %SUsed  In  Out  Read  Write
17:29:11    53.8    97.8     488.3   27.7 1139 1448  5400   5801
17:30:11    53.8    96.2     488.3   27.7 4166 3888 19079  15558
17:31:10    53.8    97.6     488.3   27.7 4222 3867 19075  15468
17:32:11    53.8      98     488.3   27.7 4324 4016 19687  16065
17:33:10    53.8    97.7     488.3   27.7 4248 3882 18738  15526
17:34:11    53.8    97.9     488.3   27.7 3472 3365 14663  13461
17:35:11    53.8    97.9     488.3   27.7 3626 3416 15288  13665
17:36:11    53.8      98     488.3   27.7 3570 3256 14933  13023
17:37:11    53.8    97.7     488.3   27.7 3598 3441 15039  13762
```

```
17:38:11   53.8      98     488.3   27.7 3696 3408 15347  13632
17:39:11   53.8    98.1     488.3   27.7 3573 3432 15044  13730
```

Here we see a strong increase in the number of pages swapped-in and swapped-out compared to our other tests. This indicates that page faults were resolved faster, leading to better performance as seen in throughput and transaction rate. The swap rate is almost double the amount seen in the DASD tests (Example 7-12).

Notice the different CPU utilization results compared to previous measurements using a real device (Example 7-6 on page 107). I/O wait time is also down, but user time is only slightly increased, and most saved I/O wait time is now reported as stolen.

Example 7-12 CPU utilization at swapping using FBA VDISK

```
Interval  <-------CPU Utilization (%)----------->
Time      TotCPU User  Kernel IOWait Idle  Stolen
17:29:11   5.5   2.3     2.7   24.4  64.9    5.2
17:30:11  14.5   5.2     7.7   68.7    0    16.8
17:31:10  14.2   5.2     7.5   69.3    0    16.5
17:32:11  14.7   5.3     7.9   68.1    0    17.2
17:33:10  14.2   5.2     7.5     69    0    16.8
17:34:11  11.8   4.3     6.3   74.5    0    13.7
17:35:11  12.2   4.5     6.5   73.6    0    14.2
17:36:11  11.9   4.4     6.3   74.5    0    13.7
17:37:11    12   4.5     6.3     74    0      14
17:38:11  12.1   4.6     6.3   73.7    0    14.3
17:39:11  12.1   4.4     6.4   73.9    0      14
```

7.4.5 Swapping to VDISK with DIAGNOSE

In this test, we swapped to VDISK with DIAGNOSE using one hogmem program, and allocating 128 MB in a 64 MB z/VM Linux guest (Example 7-13).

Example 7-13 Memory at swapping using DIAGNOSE VDISK

```
           <-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval    <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time     M_Total %MUsed  S_Total %SUsed  In  Out  Read  Write
17:49:11  53.8    97.9   488.3   27.7  1453 1831  6716   7335
17:50:10  53.8    97.7   488.3   27.7  5211 4790 23583  19159
17:51:10  53.8    97.8   488.3   27.7  5841 5518 26231  22071
17:52:11  53.8    97.7   488.3   27.7  6045 5599 26971  22395
17:53:11  53.8    97.5   488.3     28  5941 5585 28257  22356
17:54:10  53.8    96.6   488.3     28  5458 5138 25464  20555
17:55:11  53.8    97.5   488.3     28  5804 5374 26747  21499
17:56:11  53.8    97.8   488.3     28  6046 5534 27217  22138
17:57:10  53.8    97.8   488.3     28  5868 5532 26620  22129
17:58:11  53.8      98   488.3     28  5824 5417 25866  21666
17:59:11  53.8    97.2   488.3   27.9  5116 4818 22223  19274
```

The swap rate is higher when compared with VDISK using FBA. Read and write block I/O is higher than in the FBA VDISK test (Example 7-14).

Example 7-14 CPU utilization at swapping using DIAGNOSE VDISK

```
Interval  <-------CPU Utilization (%)----------->
Time      TotCPU User  Kernel IOWait Idle  Stolen
17:49:11      7    3     3.4   22.7  66.5   3.8
17:50:10   17.7  6.4     9.4   70.4    0   11.9
17:51:10   19.7  7.2    10.4   66.3    0   13.9
17:52:11   20.2  7.4    10.7   65.5    0   14.3
17:53:11   21.5  7.8    11.4     64    0   14.5
17:54:10   18.7  6.7    10.1   68.4    0   12.9
17:55:11   19.6  7.1    10.4   66.8    0   13.6
17:56:11   20.3  7.4    10.9   65.5    0   14.2
17:57:10   19.7  7.2    10.5   66.3    0     14
17:58:11   19.5  7.2    10.3   66.4    0   14.1
```

We get similar results to previous measurements for CPU utilization using VDISK (Example 7-12 on page 112). I/O wait time is down compared to FBA accessing the same VDISK. The time attributed to stolen is down and user time is up. This indicates better throughput and transaction, or better performance.

7.4.6 The advantages of a VDISK swap device

VDISK offers the advantage of allowing you to define a large swap area at relatively little expense. The VDISK is not allocated until the Linux server attempts to swap.

Enabling an FBA VDISK

DASD uses FBA by default for both SUSE and Red Hat because a VDISK appears as an FBA device to Linux. Use the **`mkswap`** command to initialize the VDISK. The **`swapon`** command cannot use the device until you complete this process. You need to issue **`mkswap`** every time that you start the virtual machine because VDISK contents are not saved after logoff (that is, VDISK data is volatile).

Choose any of these methods to initialize the VDISK:

- Modify the Linux init scripts to run **`mkswap`** in the boot process. The **`swapon`** command can then automatically pick up the disk when Linux processes the `/etc/fstab` file. You can also issue the **`swapon`** command manually if necessary.
- IPL CMS in the Linux guest, then use CMS tools to initialize the VDISK. Linux will see the VDISK as a swap device just as if **`mkswap`** was already issued. This is a good method if you also need to use CMS to couple virtual channel-to-channels (CTCs).
- Use another virtual machine to link to all the VDISKs so that the CP retains them after the Linux guest is logged off. You still need to initialize the disks after a z/VM IPL. This method forces the CP to retain additional VDISKs, which can strain the paging subsystem.

Whether you choose Linux or CMS to initialize the VDISK depends on available skills and on your willingness to modify Linux. Neither SUSE nor Red Hat currently use VDISK as the swap device, but remember that if you prepare VDISK on CMS in advance, Linux picks it up automatically.

Note: Programming VDISK in CMS can be challenging. See 7.10, “Initializing VDISK using CMS” on page 123, for an example of an initializing script for VDISK (rsrvdisk exec).

Effects of parallel swapping to VDISK

Swapping can involve using multiple swap partitions. Linux effectively spreads the I/O over multiple disks when you use multiple swap partitions with the same priority, allowing a higher I/O rate. However, this does apply to VDISKs. Because there is little queuing for VDISK, any further increase of swap space might not provide improvement.

7.5 Swapping with FCP to Linux attached SCSI disk

Linux on System z, z/VM, and System z hardware support the small Computer System Interface (SCSI) disks attached to FICON Express running in FCP mode. For this test, we used one hogmem program and allocated 128 MB in a 64 MB z/VM Linux guest. The result was higher rates for swapped-in and swapped-out pages compared to both DASD measurements, but with lower rates than when swapping to VDISK. Example 7-15 shows the test results.

Example 7-15 Memory when swapping to a SCSI disk over FCP

```
           <-Memory(MB)-> <------- Swapping -----> <-BlockIO->
Interval    <-- Main ---> <-Space (MB)> <Pgs/sec>  <--kB/sec->
Time    M_Total %MUsed  S_Total %SUsed   In   Out  Read Write
18:07:11  53.8    97.1   3815    3.5  839.6  1188  4483  4764
18:08:11  53.8    98.0   3815    3.5   2789  2690 12655 10760
18:09:10  53.8    98.2   3815    3.5   2681  2509 12080 10036
18:10:10  53.8    98.1   3815    3.5   2949  2823 13846 11293
18:11:10  53.8    96.7   3815    3.5   2966  2782 13720 11128
18:12:10  53.8    98.0   3815    3.5   2877  2592 13071 10370
18:13:10  53.8    97.8   3815    3.5   2842  2666 13148 10666
18:14:11  53.8    97.9   3815    3.5   2876  2707 13038 10829
18:15:11  53.8    97.4   3815    3.5   2816  2656 12871 10628
18:16:10  53.8    97.7   3815    3.5   2844  2688 13117 10751
18:17:10  53.8    97.3   3815    3.5   2759  2647 12593 10589
```

Example 7-16 shows less I/O wait time than in the DASD tests. More is attributed to user time, which improves the application throughput and transaction rate. Also, more stolen time is reported than in the DASD tests. The SCSI characteristics are better than in DASD, with little time for involuntary wait (stolen).

Example 7-16 CPU utilization swapping to SCSI disk over FCP

```
Interval  <-------CPU Utilization (%)----------->
Time      TotCPU User  Kernel IOWait Idle  Stolen
18:08:11    11.1  4.2    5.0   88.9    0     0.1
18:09:10    10.6  4.0    4.8   89.3    0     0.1
18:10:10    12.0  4.4    5.6   87.9    0     0.1
18:11:10    11.9  4.4    5.6   88.0    0     0.1
18:12:10    11.1  4.3    5.0   88.8    0     0.1
18:13:10    11.4  4.2    5.3   88.5    0     0.1
18:14:11    11.2  4.3    5.1   88.7    0     0.1
18:15:11    11.1  4.2    5.1   88.8    0     0.1
18:16:10    11.4  4.2    5.4   88.5    0     0.1
18:17:10    11.0  4.1    5.1   88.9    0     0.1
```

7.6 Swapping to DCSS

You can also define swap space in DCSS by loading DCSS as a block device in Linux with the dcssblk device driver. Use the special multipart EW/EN type of DCSS for swapping. In this DCSS type, only the first 4 KB page is actually saved on z/VM spool space, or the EW the exclusive-writable part of the DCSS. That first 4 KB contains the swap signature of a swap device. The remaining part of the DCSS is defined as EN (that is, exclusive-nonsaved), which means that it is not backed up on the z/VM spool space.

This special type of DCSS loads faster than a regularly backed DCSS. In this configuration, you need to define only one DCSS for swapping because defining it as an exclusive type means that the same DCSS can be loaded from multiple guests, and that each guest gets its own exclusive content. Example 7-17 shows what CMS commands define and save a DCSS of 1 GB size, starting at address 1 GB.

Example 7-17 Define and save DCSS

```
defseg swapping 40000-40000 ew 40001-7ffff en
saveseg swapping
```

Using DCSS for swapping gives you the following advantages:

- Offers fast write into z/VM's storage, and swap caching when the Linux guest is memory constrained but z/VM is not.
- You can shrink guest virtual memory size while maintaining acceptable performance for peak workloads.
- Requires no channel programs like VDISK access. Data transfers are performed only by z/VM memory management.
- You can realize a low SIE break-rate.

DCSS swapping is currently the fastest known method. However, it can add complexity to your system, so we recommend that you use it as small fast swap device in peak situations. The DCSS swap device should be the first in a cascade of swap devices, where subsequent devices would be bigger and slower, for exmaple, real disk. We did not conduct tests to measure swapping to DCSS.

7.7 Comparing swap rates of tested swap options

In this section, we compare all the measurements of swap rates and the CPU consumption of the options that we tested (Figure 7-3). The swap rate is a performance measure of how fast pages can be transferred to or from the swap device. Swapped-in and swapped-out rates are added to an overall swap rate. The measurements show the Linux point of view, so remember that this experiment is running on top of the z/VM virtualization layer.

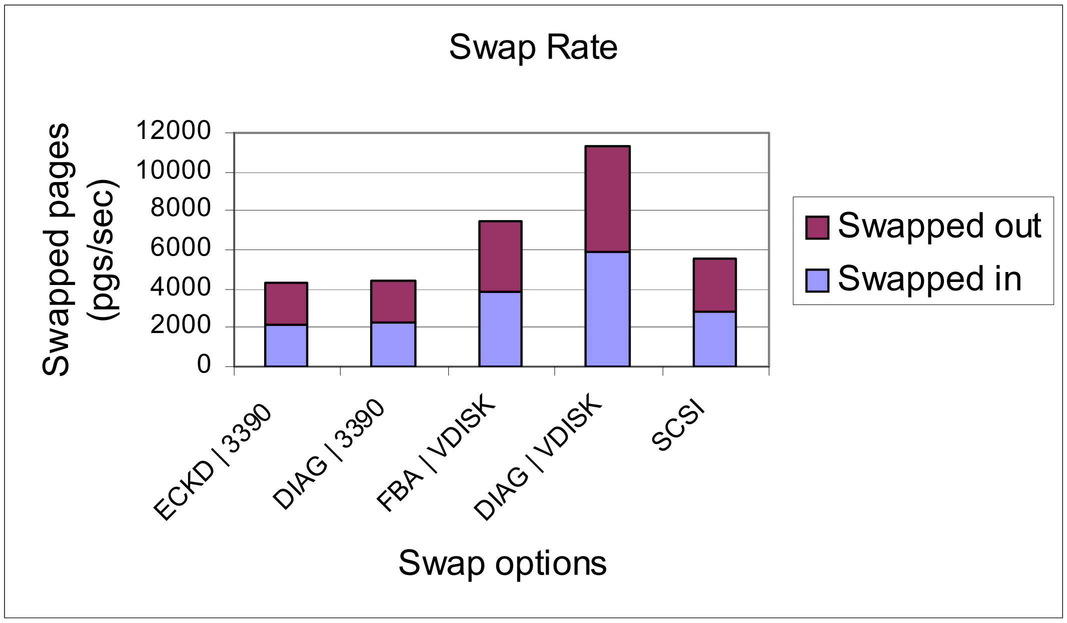


Figure 7-3 Comparison of swap options

We did not see much difference in swapping rate to DASD using ECKD or DIAGNOSE. The SCSI disk produced better performance, but VDISK using FBA was faster. VDISK using DIAGNOSE access method was the fastest option that we tested, which we expected because VDISK resides in memory. Deciding whether VDISK is the best option depends on both performance and the scenario. If the environment is short on memory, VDISK is not the best choice.

Figure 7-4 compares total CPU and CPU consumption details. We ran the test over a constant time rather than based on completing a specific block of work. If we used a fixed block of work as a goal, the test could have finished earlier if the page allocation request was satisfied faster.

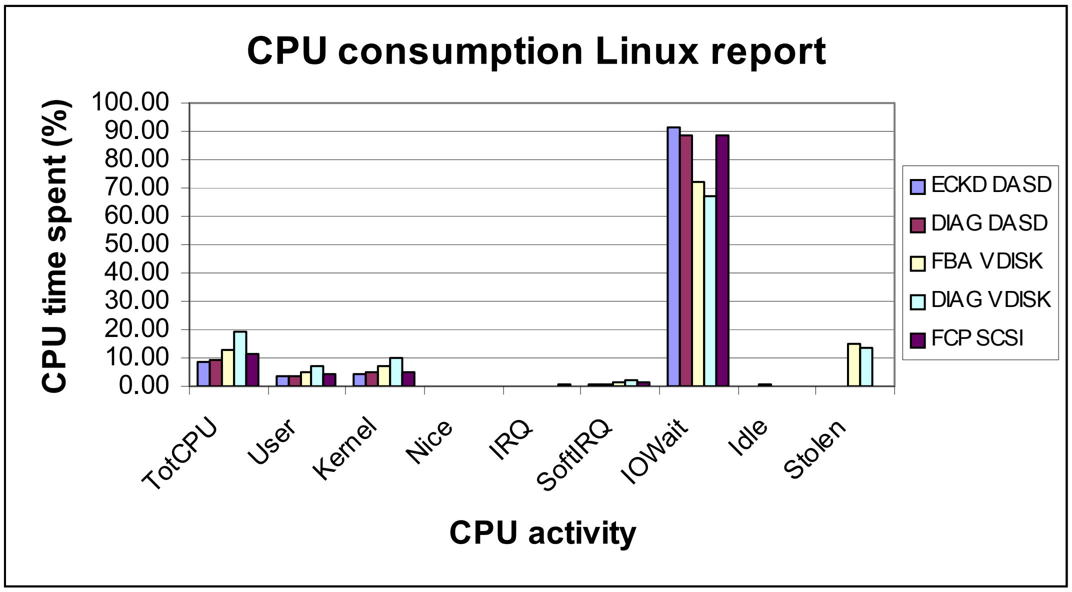


Figure 7-4 Linux CPU consumption report for swap options

As expected, most of the time is reported to IOWait. I/O going out to physical devices (that is, not satisfied by caches) is expected to be slower than other operations residing in memory. We see lower values for the VDISK tests counted to IOWait. Stolen time is reported for VDISK only. The highest Total CPU (TotCPU) and kernel CPU usage is reported for VDISK. For the application performance, the user CPU is most important because it shows the highest values for combinations DIAGNOSE VDISK and FBA VDISK. Note that the results are not in ratio to the swapped pages, so faster options swap more in a specific time and need more CPU time. This explains the high TotCPU values for the VDISK options.

Comparing costs seen by z/VM

We used the z/VM Performance Toolkit to illustrate the CPU costs as seen by z/VM (Figure 7-5). The costs counted on System CPU were constant throughout the various tests. The VDISK experiments, bar 3 and 4 from the left, have far higher CP supervisor user CPU values than all of the other options. But VDISK tests complete a given work unit faster, and our test ran over a specified elapsed time and was not based on finishing a specified block of work. This proves that faster VDISK swapping needs relatively more overall CPU while it is running. SCSI was faster than the ECKD DASD and DIAGNOSE DASD swapping and needed fewer USER CPU resources.

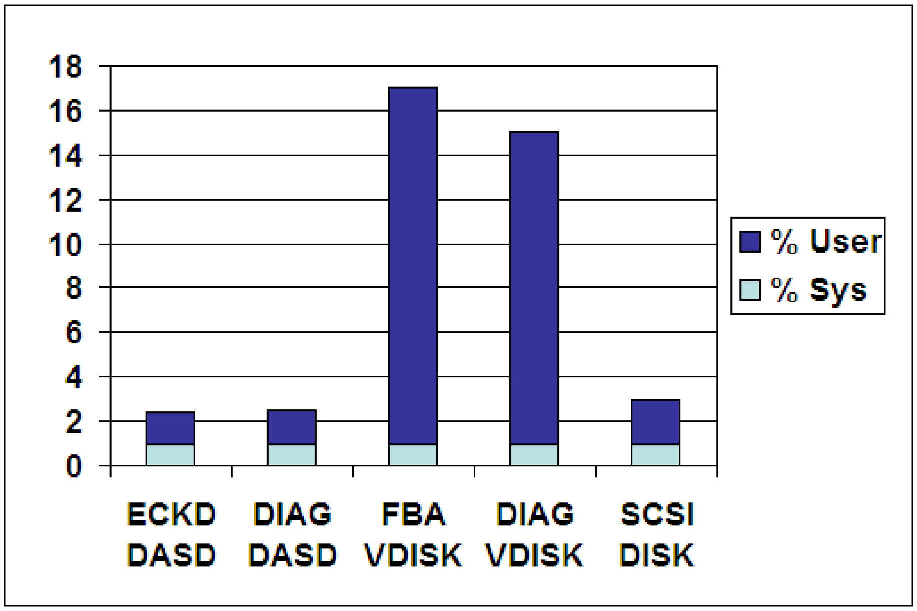


Figure 7-5 CPU costs as reported by z/VM

7.8 Swapping recommendations

The worst-case scenario for your system is when applications need more memory and the request cannot be satisfied. This could be caused by newly introduced bad program code, memory leak, or the application starting a large number of threads. In this case, the Linux kernel kills processes using the OOM-killer Linux kernel function. We label this as the worst case because it leads to abnormal application ends and potential data loss. It is better to swap to slow devices and reduce performance than to lose data and end programs abnormally.

Make sure to plan swapping for your system. If you plan to use swapping as an exception, make sure that your swap devices are high performance. This means that, in z/VM, you must define VDISK as the swap device with the highest priority. Because VDISK resides in main memory, we do not recommend using it if your main memory is a constrained resource. Swapping on a regular base is observed at higher resource overcommit rates, as often seen in low-utilized, consolidated environments:

- 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 2.3, “Sizing considerations for z/VM Linux guests” on page 11, reduce the virtual machine size of the 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 does not apply to a z/VM Linux guest. If a Linux guest uses this much swap space, it probably indicates that a larger virtual machine size should be allocated to the guest.

- Do not enable MDC on Linux swap minidisks. The read ratio is not high enough to overcome the write overhead.
- Swapping to VDISK with the DIAGNOSE access method is faster than swapping to DASD or SCSI disk. Also, with a VDISK swap device, your z/VM performance management product can report swapping by a Linux guest.
- 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 7.4.4, “Swapping to VDISK with FBA” on page 111, 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.

- Select swap devices over swap files.

  Linux on System z is able to use swap files instead of swap partitions or swap disks. The swap files introduce extra effort to handle the file system.

7.8.1 Cascaded swap devices

There good reasons to use multiple swap disk devices with different priorities. In such a configuration, Linux fills the device with the highest priority before using the next one. The algorithms Linux uses to allocate pages on swap devices cause the active area to sweep over the device. This reduces the device seek times and increases Linux’s ability to build long I/O chains. This means that over time the entire swap device has been referenced.

When using VDISK, the CP must provide memory for the entire VDISK, even though Linux might only need a small portion VDISK at any given time. If memory contention is high enough, the contents of the VDISK are paged out by CP and must be brought back in when Linux next references that part of VDISK.

You can reduce the footprint by 50% by giving the Linux virtual machine two smaller VDISKs instead of one big VDISK and then using them as swap devices with different priorities.

We recommend using a smaller VDISK with higher priority and a larger disk. That is, DASD or SCSI, with a lower priority in cases where you expect swapping in peak situations. If you always expect paging on your system, using VDISK is not an advantage because it reduces available memory.

7.8.2 Page-cluster value impact

The page-cluster value determines how many additional pages Linux will swap in on a page fault, assuming that the process will also request the next pages. A value of 1 translates in this context to 2 (2 power 1). The default value for the page-cluster size is 3, which makes that the unit seen as 8 pages (2 power 3). The /proc/sys/vm/page-cluster pseudo-variable controls page-clustering.

Example 7-19 shows page-clustering effects. We used hogmem. See 7.9, “hogmem program text” on page 122, for details. We started the tests at 16:29 and set the page-cluster size to 1 before starting:

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

We set the page-cluster value to 3 between 16:34:33 and 16:35:33 (default value on SLES10 SP1):

```
# echo 3 > /proc/sys/vm/page-cluster
```

Swap activity as reported by **`vmstat`** showed no significant change during this period for swapped-in and swapped-out pages. However, it did show changes in I/O for blocks in and blocks out, more interrupts, and decreased context switches (Example 7-18).

Example 7-18 Changed page-cluster value observed by vmstat

```
g73vm1:~ # vmstat 60
procs -----------memory---------- ---swap-- -----io---- -system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  2 139604   1156     84   1584 5955 5599  6638  5599  855 1563  2  4  0 93  0
 0  1 139604   1712     48    384 6306 6554  6851  6559  877 1652  3  5  0 93  0
 0  2 139604   2168     48   1368 5618 5491  5992  5492  788 1472  2  4  0 94  0
 0  1 139604   1608     44    844 5868 5969  6198  5969  814 1534  2  4  0 94  0
 0  2 139604   1096     44    608 5960 5525  6451  5525  842 1557  2  4  0 93  0
 0  2 139604   1340     60   4344 6187 6124  6640  6124  870 1614  2  4  0 93  0
 0  3 139604   1808     48    816 7234 7122  7633  7122  498  868  3  4  0 93  0
 0  1 139604   1360     48    940 8586 7792  9086  7792  396  640  3  4  0 92  0
 0  1 139604   1536     48     96 8186 8112  8622  8112  384  610  3  4  0 93  0
 0  1 139604   1420     48    304 8602 8033  9105  8033  380  641  3  4  0 92  0
 0  3 139604   1312     48    904 7851 7373  8308  7373  372  594  3  4  0 93  0
 0  1 139604   1104     48    796 8427 7762  8988  7762  394  636  3  4  0 93  0
 0  2 139604   1068     48    536 8261 7438  8714  7438  420  622  3  4  0 93  0
```

In Example 7-19, we used z/VM Performance Toolkit to show test results from the Linux point of view.

Example 7-19 Changed page-cluster value observed by z/VM Performance Toolkit

```
 FCX162      CPU 2084  SER F80CA  Interval 01:53:46 - 16:44:33    Perf. Monitor

 Resource Usage Log for User G73VM1

           <----- CPU Load -----> <------ Virtual IO/s ------>
 Interval       <-Seconds->   T/V
 End Time  %CPU  TCPU  VCPU Ratio Total DASD Avoid Diag98   UR Pg/s  User Status
 >>Mean>>   ...  ....  ....   ...   ...  ...   ...    ...  ...  ...  ---,---,---
 16:29:33  8.80 5.279 4.352   1.2   828  828    .0     .0   .0   .0  EME,CLO,DIS
 16:30:33  9.05 5.432 4.469   1.2   850  850    .0     .0   .0   .0  EME,CLO,DIS
```

```
16:31:33  8.22 4.933 4.059   1.2   781  781    .0     .0   .0   .0  EME,CLO,DIS
16:32:33  7.83 4.700 3.868   1.2   751  751    .0     .0   .0   .0  EME,CLO,DIS
16:33:33  8.38 5.025 4.118   1.2   790  790    .0     .0   .0   .0  EME,CLO,DIS
16:34:33  8.85 5.308 4.356   1.2   836  836    .0     .0   .0   .0  EME,CLO,DIS
16:35:33  8.16 4.896 4.150   1.2   562  562    .0     .0   .0   .0  EME,CLO,DIS
16:36:33  8.50 5.099 4.467   1.1   345  345    .0     .0   .0   .0  EME,CLO,DIS
16:37:33  8.68 5.208 4.568   1.1   342  342    .0     .0   .0   .0  EME,CLO,DIS
16:38:33  8.99 5.396 4.730   1.1   359  359    .0     .0   .0   .0  EME,CLO,DIS
16:39:33  8.25 4.949 4.340   1.1   340  340    .0     .0   .0   .0  EME,CLO,DIS
16:40:33  8.59 5.151 4.514   1.1   354  354    .0     .0   .0   .0  EME,CLO,DIS
16:41:33  8.69 5.212 4.566   1.1   352  352    .0     .0   .0   .0  EME,CLO,DIS
16:42:33  8.56 5.134 4.490   1.1   363  363    .0     .0   .0   .0  EME,CLO,DIS
```

Note the following points in Example 7-19 on page 120:

- A higher DASD I/O rate in the interval 16:29:33 to 16:34:33. The page-cluster value in this interval is set to 1.
- The DASD I/O rate decreases dramatically at 16:34:33/16:35:33. At the beginning of this interval, we returned the page-cluster value to the default of 3.
- This test shows that a small page_cluster value causes many short I/Os.

In Figure 7-6, we show the real DASD I/O measured by the z/VM Performance Toolkit. Note that the number of I/Os decreases after we set /proc/sys/vm/page-cluster back to the default of 3. Using a page_cluster size of 1 speeds up swapping with the hogmem test program but increases the number of interrupts and context switches in Linux. It also increases the virtual and real DASD I/O reported in z/VM

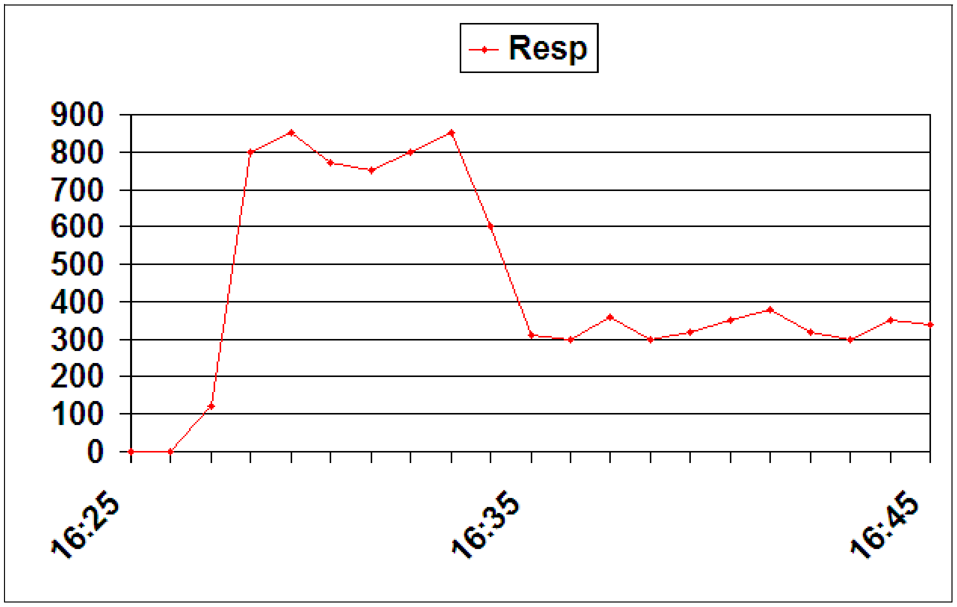


Figure 7-6   Real DASD I/O for swap disk DASD 4443

Note: The test results differ from our expectations and from our former recommendations to use a page-cluster size of 1. The reason is that the hogmem test program touches pages walking linearly through the memory. The result is that larger units of page clusters that where swapped in at once include the pages needed next. If hogmem would touch pages randomly, the swapping of eight pages at once would bring in many pages that where not needed.

7.9 hogmem program text

hogmem allows you to run processes that allocate and use virtual memory. We used hogmem to compare efficiency of the various swap devices in Linux. Example 7-20 shows the hogmem program listing. The program also reports memory bandwidth in MBps, but this generally has little value for Linux.

Example 7-20 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)

#ifndef CLK_TCK
# define CLK_TCK CLOCKS_PER_SEC
#endif

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);
}
```

7.10 Initializing VDISK using CMS

Use the RSRVDISK EXEC (Example 7-21) to initialize a VDISK in CMS before starting Linux. Our example shows sample code that you can use as a guideline. Customize this script and then call it from PROFILE EXEC.

Example 7-21 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 use **mkswap** to initialize the disk. This does not undo the CMS formatting, but it causes Linux to write blocks in the disk payload (that is, the single CMS file created on the disk). Next, shut down the Linux guest and IPL CMS, without running the **swapon** command. Example 7-22 shows you how to copy the modified blocks.

Example 7-22 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
```

Use the **pipe** command (Example 7-23) to create a file on the A disk to hold the modified blocks. Note that it shows that only a single block was modified. You can use this file to prepare a fresh VDISK.

Example 7-23 Prepare a fresh VDISK

```
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
```

Linux now sees a new VDISK the same as the one previously prepared with the **mkswap** command. Examine the SAMPLE SWAP0207 file contents and you will see that it is not difficult to create contents and make them effective for VDISKs of any size.

You can use a similar process if you want VDISK to hold temporary files for Linux. In this case, use the **mke2fs** command instead of **mkswap** against the device before you copy the payload. You can create files and directories, if necessary, to have the disk in the correct state when Linux boots up.

8

CPU resources, LPARs, and the z/VM scheduler

In this chapter we discuss processor resources, LPARs in the z/Linux environment, and the z/VM scheduler. We look at LPAR weights and options and learn about the z/VM 6.1 and CP Scheduler. We also examine SRM controls, CP SHARE, CPU time accounting, and the Virtual Machine Resource Manager.

8.1 Understanding LPAR weights and options

zEnterprise systems have extensive virtualization capabilities inherent in the hardware. Each virtualized operating system runs in a logical partition (LPAR). One LPAR with z/VM can accommodate hundreds, even thousands, of guests regardless of whether they are z/VM, Linux, z/OS, or zVSE guests. Even so, it might be better to divide processor resources into several LPARs instead of running one LPAR and a single z/VM environment. Here are two reasons to divide the processor:

- To dedicate processors to specific environments for accounting purposes (for example, for corporate departments, governmental agencies, or specific major)
- To dedicate processors and storage to mission-critical systems, thus ensuring that the system is responsive and available

8.1.1 LPAR configuration considerations

Your goal is most likely to configure your system to best suit your business requirements. Understanding how LPAR is configured across engines and how LPAR options work and perform will help you achieve that goal. In this section, we discuss LPAR configuration options.

As you design your system, you first need to determine what engines will be used in the LPAR and whether the LPAR will span different types of engines. This will help you create a system that will deliver the performance and mission-critical tasks your business needs.

zLinux LPAR configuration can use five types of engines, or processor types:

- Central processor (CP)
- Internal Coupling Facility Processor (ICF)
- Integrated Facility for Linux Processor (IFL)
- IBM System z Integrated Information Processor (zIIP)
- IBM System z Application Assist Processor(zAAP)

Each engine type has characteristics that will affect system performance. For example, IFLs run at a higher speeds than CP engines because CP engines are affected by the multiprocessor factor, which is the efficiency losses while managing multiple processors. You also need to consider whether the work can take advantage of the specialty engines, zIIPs, and zAAPs. Then you need to define the CPU Affinity option. When CPU Affinity is OFF, work is dispatched only to the CP engines even if the work is suitable for specialty engines. If CPU Affinity is ON, work is dispatched to the respective engine type.

You need to examine LPAR weighting, capping, and dedicated engines, and consider how they affect how the CP Scheduler dispatches to various engine types. We suggest that you not mix engine types within a LPAR because doing so can have unforeseen consequences in determining priorities for mission-critical workloads.

The VM Performance Toolkit Report, FCX126 Logical Partition Activity report is a robust and detailed record that can help you understand your system configuration. Figure 8-4 on page 130 displays the report. Figure 8-4 on page 130 shows the bottom section of the report. Use the Forw button to scroll through the report. This report section shows system configuration details, for example, 12 CP engines, 2 zAAP engines, 2 IFL engines, 6 ICF engines, and 2 zIIP engines, for a total of 24 engines. It also shows how the engines are weighted in the configuration. These details help you understand the data contained in the rest of the report.

```
General LPAR mgmt overhead                    .9
 Overall physical load                       13.5

 Summary of physical processors:
 Type  Number  Weight  Dedicated    %LPBUSY    %LPOVHD    %NCOVHD      %BUSY
 CP        12     440          2         93          4         15        112
 ZAAP       2      20          0          0          0          0          0
 IFL        2      10          0          5          0          1          6
 ICF        6      10          2        200          0          5        205
 ZIIP       2      20          0          0          0          0          0
```

Figure 8-1 FCX126 Logical Partition Activity Report (LPAR)

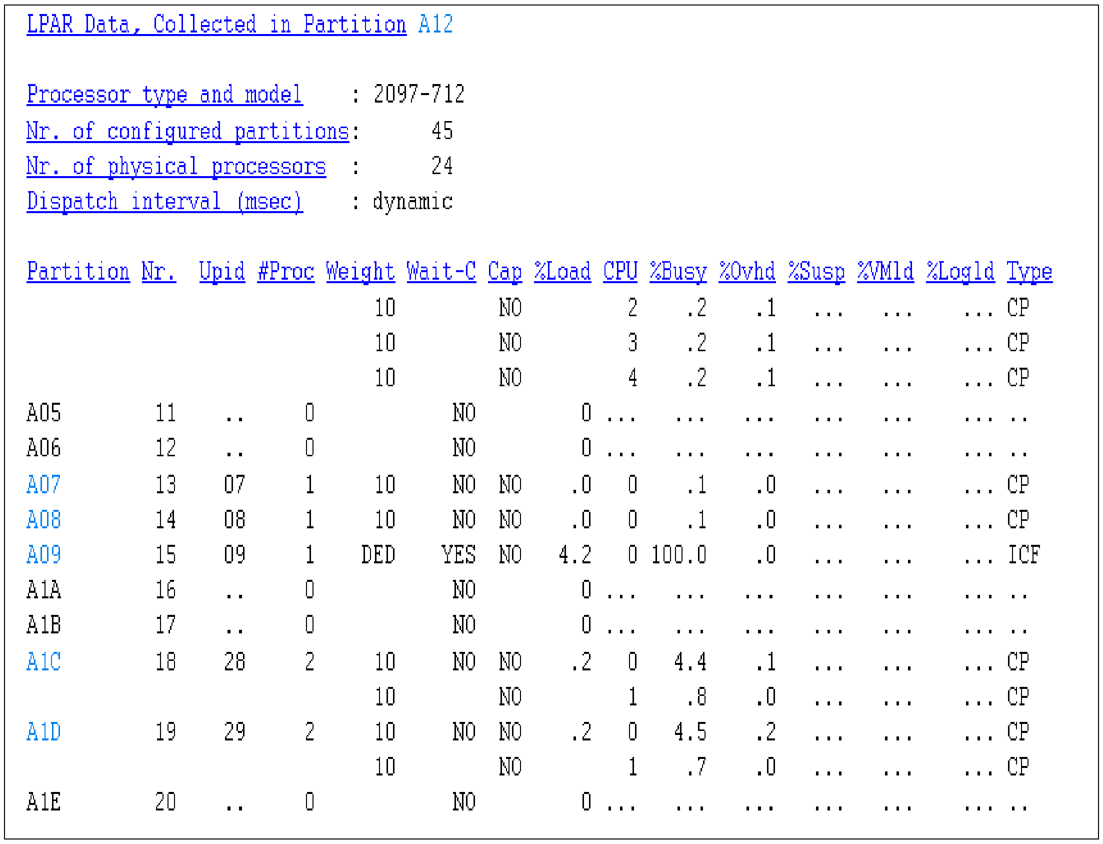
Figure 8-2 shows the beginning of the report. This section details the machine type, the number of partitions configured, and the number of processors, in addition to this information:

- Processor type and model
- Nr. of configured partitions: Total number of partitions on the entire system
- Nr. of physical processors: Total number of physical processors installed and enabled
- Dispatch interval, which is set to dynamic

```
 Processor type and model    : 2097-712
 Nr. of configured partitions:      45
 Nr. of physical processors  :      24
 Dispatch interval (msec)    : dynamic
```

Figure 8-2 FCX126 Logical Partition Activity Report (LPAR)

We now look at the FXC126 Logical Partition Activity Report produced with the VM Performance Toolkit (Figure 8-3).



```
LPAR Data, Collected in Partition A12

Processor type and model    : 2097-712
Nr. of configured partitions:      45
Nr. of physical processors  :      24
Dispatch interval (msec)    : dynamic

Partition Nr.  Upid #Proc Weight Wait-C Cap %Load CPU %Busy %Ovhd %Susp %VMld %Logld Type
                              10         NO         2    .2    .1   ...   ...    ... CP
                              10         NO         3    .2    .1   ...   ...    ... CP
                              10         NO         4    .2    .1   ...   ...    ... CP
A05        11    ..     0            NO         0 ...   ...   ...   ...   ...    ... ..
A06        12    ..     0            NO         0 ...   ...   ...   ...   ...    ... ..
A07        13    07     1     10     NO  NO    .0   0    .1    .0   ...   ...    ... CP
A08        14    08     1     10     NO  NO    .0   0    .1    .0   ...   ...    ... CP
A09        15    09     1    DED    YES  NO   4.2   0 100.0    .0   ...   ...    ... ICF
A1A        16    ..     0            NO         0 ...   ...   ...   ...   ...    ... ..
A1B        17    ..     0            NO         0 ...   ...   ...   ...   ...    ... ..
A1C        18    28     2     10     NO  NO    .2   0   4.4    .1   ...   ...    ... CP
                              10         NO         1    .8    .0   ...   ...    ... CP
A1D        19    29     2     10     NO  NO    .2   0   4.5    .2   ...   ...    ... CP
                              10         NO         1    .7    .0   ...   ...    ... CP
A1E        20    ..     0            NO         0 ...   ...   ...   ...   ...    ... ..
```

Figure 8-3 FCX126 LPAR Activity Report

This report shows what z/VM considers as its processor utilization for each LPAR and the total for all partitions in the entire System Z server machine. The %Busy represents the time that a physical processor is assigned to a logical processor, and includes overhead. The %Ovhd accounts for only LPAR management. It is important that you note the details of what each engine is doing, which engines are busy, and which are not. These facts will help identify runaway, over-burdened, under-used, or stalled partitions. LPARs are prioritized by weights, which are explained in 8.1.3, “Converting weights to logical processor speed” on page 134. %Busy and %Ovhd percentages are both out of 100%, where 100% means one physical engine’s worth. Thus, a value of 100% in %Busy indicates that the engine is part of a dedicated partition, or that it is completely busy.

Table 8-1 lists the field descriptions in the FCX126 LPAR Activity report.

Table 8-1 FCX126 LPAR field descriptions

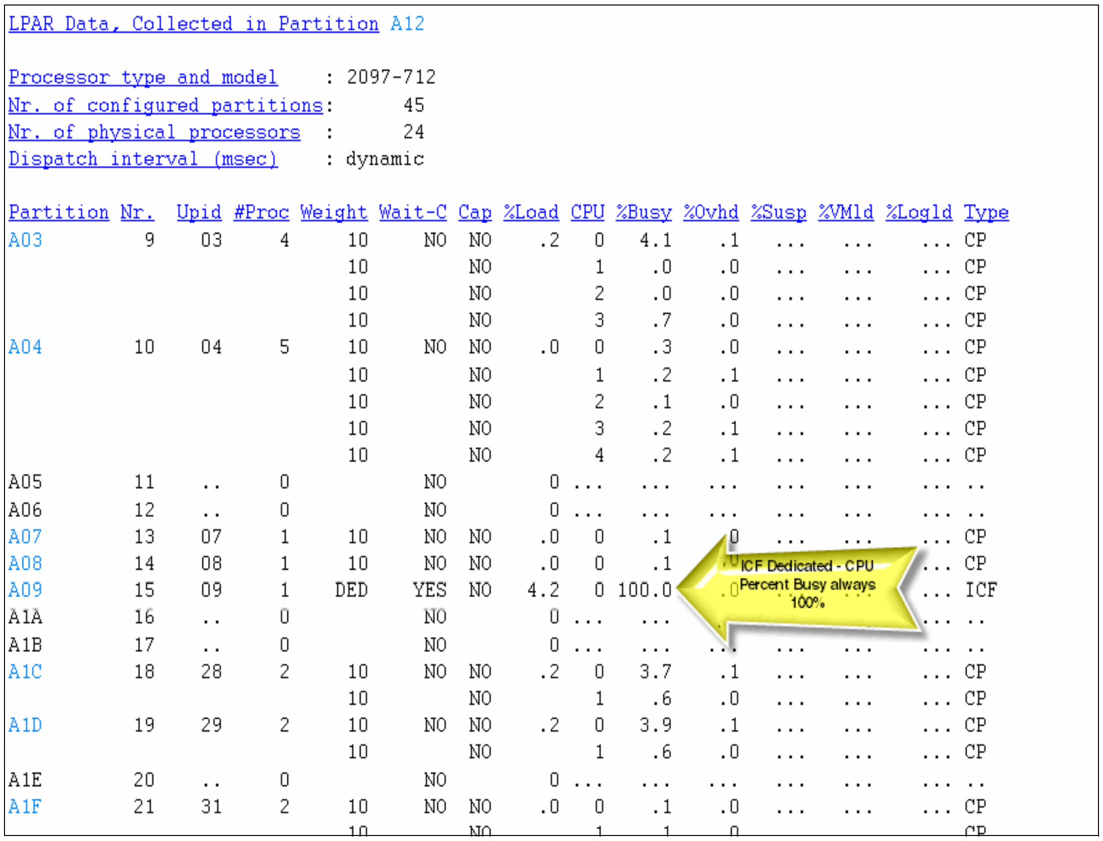
Report field	Description
Partition	Name of the logical partition.
Nr.	Number of the logical partition.
Upid	User partition ID for the logical partition.
#Proc	Total number of processors defined for the partition.

Report field	Description
Weight	Determines the share of time that the partition is granted control of the processors, based on engine type. If an engine is dedicated to the partition, DED will replace the numerical value.
Wait completion	Determines whether the logical processor keeps running on the real processor until the allocated time slice completes, even if it has no work to execute and is waiting. Set this option to YES. Set this option to NO to give control to other processors before time slice completes, when the real processor is waiting and no longer executing.
Cap	YES indicates that the CPU is provided up to the allocated maximum only. NO indicates that the CPU is provided up to the maximum and beyond if available.
% Load	Relative load of this partition, based on the time its logical processors were active, divided by the totally available processor time, and expressed as a percentage.
CPU	CPU identification character allocated to each processor in the partition, from 0 to the total number of CPUs assigned.
% Busy	Percentage of time the logical processor was busy and assigned to a real processor.
% Ovhd	Percentage of elapsed time spent to manage LPAR, the overhead for running and controlling the various workload partitions executing under the LPAR control.
%Susp	Percentage of time the logical processor was suspended and could not provide service to the guest system due to LPAR management time and contention for real processors. This can occur for these reasons: ▶ The logical processor was ready to run but could not find a place to run because all physical matching types were busy. In this case, you might need a bigger CEC. ▶ The logical processor did something that induced overhead. This %Ovhd time is accruing. ▶ The logical processor invoked a function that resulted in the logical processor becoming undispatchable. For example, the z/VM spin lock manager might be busy. ▶ The logical processor is running in a capped partition, and the partition has used all of its entitlements. The system has stopped the partition from running, but will start it again at the next time slice.
%VMld	The VM load in a percent of a physical engine that z/VM believes it achieved on the logical processor. A value of 100% indicates that z/VM was able to use that logical processor to consume one physical engine's worth of power. This represents the VM guest time, Control Program time, and non-chargeable Control Program time. Note that %VMld is very close to the difference between %Busy and %Ovhd.
%Logld	Percent of a z/VM-accounted time that is not z/VM wait time. %Logld is not an expression of physical processor consumption, but represents the fraction of the z/VM-accounted time that is not z/VM accounted wait time.

Report field	Description
Type	CPU type of the logical processors defined for the partition. The types listed below are available only on z/VM V5.3 and later. For earlier releases, this field shows CP only, regardless of the actual engine type. Processor types: ► CP: Central processor ► ICF: Internal Coupling Facility Processor ► IFL: Integrated Facility for Linux Processor ► ZIIP: IBM System z Integrated Information Processor (zIIP) ► ZAAP: IBMystem z Application Assist Processor(zAAP)

Several of these fields are discussed further in 8.1.5, “LPAR options” on page 135.

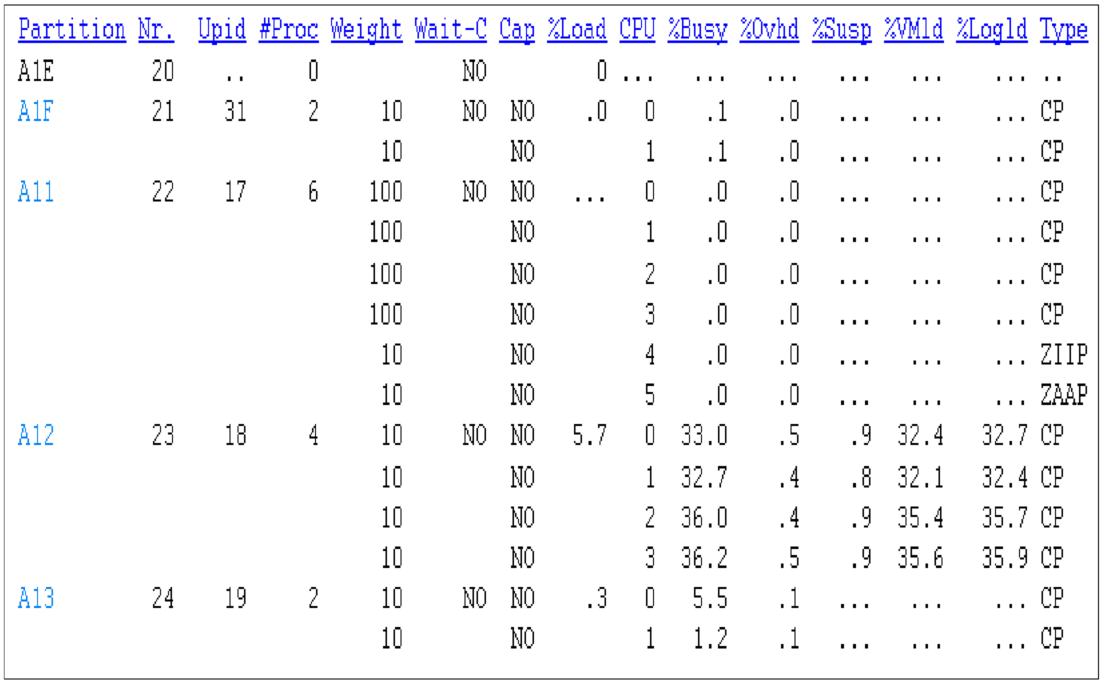
Figure 8-4 shows how a dedicated engine is represented in the report. Here, partition A09, Nr. 15 has one ICF engine dedicated to it. This indicates that it is the only partition able to use this engine.



```
LPAR Data, Collected in Partition A12

Processor type and model      : 2097-712
Nr. of configured partitions:       45
Nr. of physical processors    :       24
Dispatch interval (msec)      : dynamic

Partition Nr.  Upid #Proc Weight Wait-C Cap %Load CPU %Busy %Ovhd %Susp %VMld %Logld Type
A03          9   03     4     10     NO  NO    .2   0   4.1    .1   ...   ...    ... CP
                              10         NO         1    .0    .0   ...   ...    ... CP
                              10         NO         2    .0    .0   ...   ...    ... CP
                              10         NO         3    .7    .0   ...   ...    ... CP
A04         10   04     5     10     NO  NO    .0   0    .3    .0   ...   ...    ... CP
                              10         NO         1    .2    .1   ...   ...    ... CP
                              10         NO         2    .1    .0   ...   ...    ... CP
                              10         NO         3    .2    .1   ...   ...    ... CP
                              10         NO         4    .2    .1   ...   ...    ... CP
A05         11   ..     0            NO             0 ...   ...   ...   ...   ...    ... ..
A06         12   ..     0            NO             0 ...   ...   ...   ...   ...    ... ..
A07         13   07     1     10     NO  NO    .0   0    .1    .0   ...   ...    ... CP
A08         14   08     1     10     NO  NO    .0   0    .1    .0                 ... CP
A09         15   09     1    DED    YES  NO   4.2   0 100.0    .0                 ... ICF
A1A         16   ..     0            NO             0 ...   ...                    ... ..
A1B         17   ..     0            NO             0 ...   ...   ...              ... ..
A1C         18   28     2     10     NO  NO    .2   0   3.7    .1   ...   ...    ... CP
                              10         NO         1    .6    .0   ...   ...    ... CP
A1D         19   29     2     10     NO  NO    .2   0   3.9    .1   ...   ...    ... CP
                              10         NO         1    .6    .0   ...   ...    ... CP
A1E         20   ..     0            NO             0 ...   ...   ...   ...   ...    ... ..
A1F         21   31     2     10     NO  NO    .0   0    .1    .0   ...   ...    ... CP
                              10         NO         1    .1    .0                    CP
```

Figure 8-4   FCX126 Logical Partition Activity - Example of a Dedicated ICF Engine

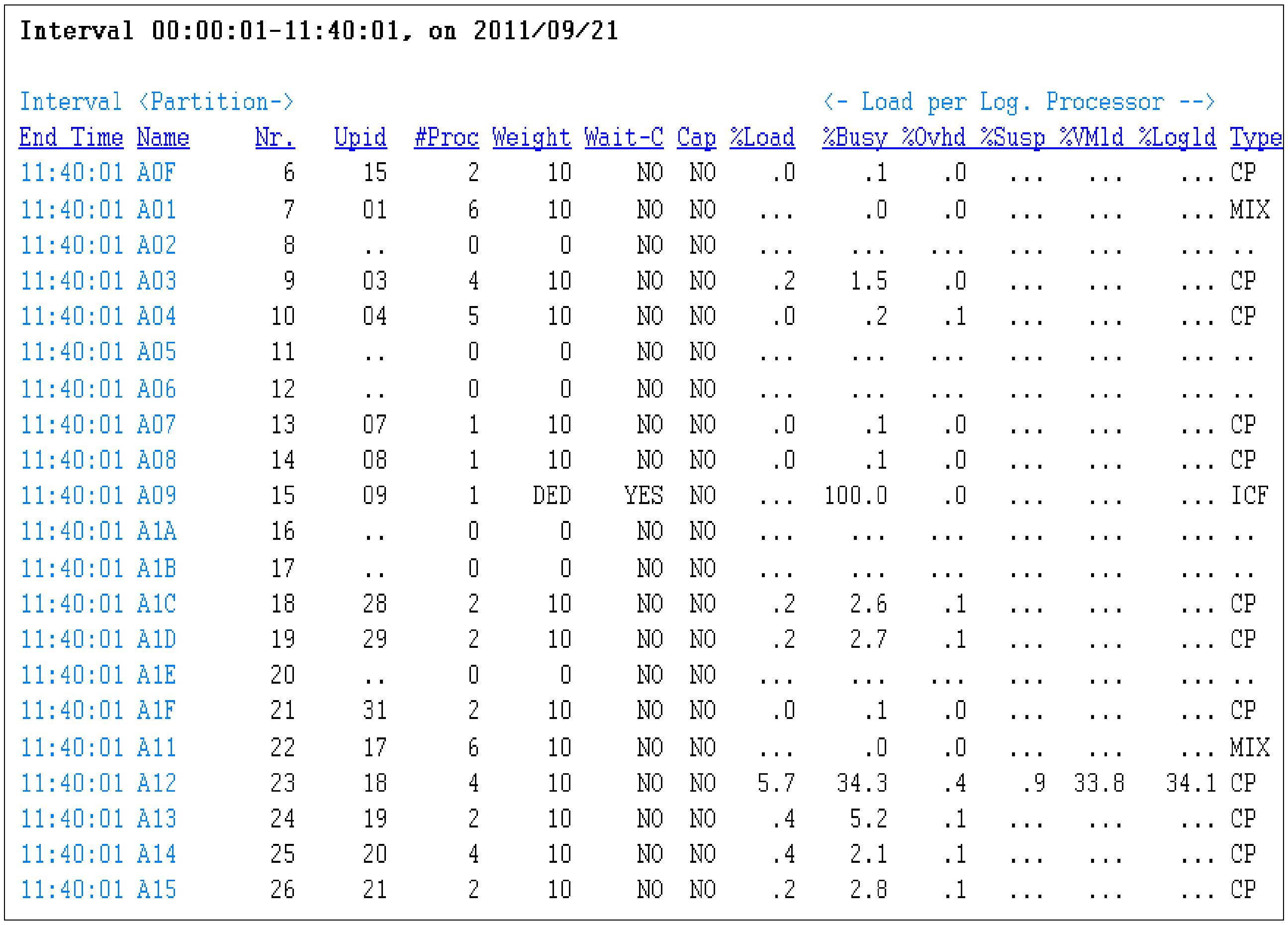
Now we consider how busy a single partition is within the LPAR by looking at a single partition, A12 (Figure 8-5). This partition is on four CP engines, is active, and is not dedicated. We add up the %Busy for all engines and then take the average to determine how busy the partition is. Here, because we have four engines, it is 137.9/400, or 34.4%. The overhead is .4% (.18/4), and the suspend time is .875% (3.5/4).



```
Partition Nr.  Upid #Proc Weight Wait-C Cap %Load CPU %Busy %Ovhd %Susp %VMld %Logld Type
A1E        20    ..     0           NO           0 ...   ...   ...   ...   ...    ... ..
A1F        21    31     2     10    NO  NO      .0   0    .1    .0   ...   ...    ... CP
                              10        NO           1    .1    .0   ...   ...    ... CP
A11        22    17     6    100    NO  NO     ...   0    .0    .0   ...   ...    ... CP
                             100        NO           1    .0    .0   ...   ...    ... CP
                             100        NO           2    .0    .0   ...   ...    ... CP
                             100        NO           3    .0    .0   ...   ...    ... CP
                              10        NO           4    .0    .0   ...   ...    ... ZIIP
                              10        NO           5    .0    .0   ...   ...    ... ZAAP
A12        23    18     4     10    NO  NO     5.7   0  33.0    .5    .9  32.4   32.7 CP
                              10        NO           1  32.7    .4    .8  32.1   32.4 CP
                              10        NO           2  36.0    .4    .9  35.4   35.7 CP
                              10        NO           3  36.2    .5    .9  35.6   35.9 CP
A13        24    19     2     10    NO  NO      .3   0   5.5    .1   ...   ...    ... CP
                              10        NO           1   1.2    .1   ...   ...    ... CP
```

Figure 8-5 FCX126 Logical Partition Activity (LPAR)

Another report is the FCX202 LPAR Log (Short name: LPARLOG) that shows a more concise view of the LPAR environment (Figure 8-6). This report is a time-indexed, interval-by-interval summary of the CPU time charged to each partition, rolled up by partition instead of broken down as in the LPAR Activity Report. This is a very helpful report because you can see, at a glance, which partitions are consuming most of the resources. For each partition, the LPARLOG adds up the utilization for the partitions and then divides to compute the average utilization for the partition.



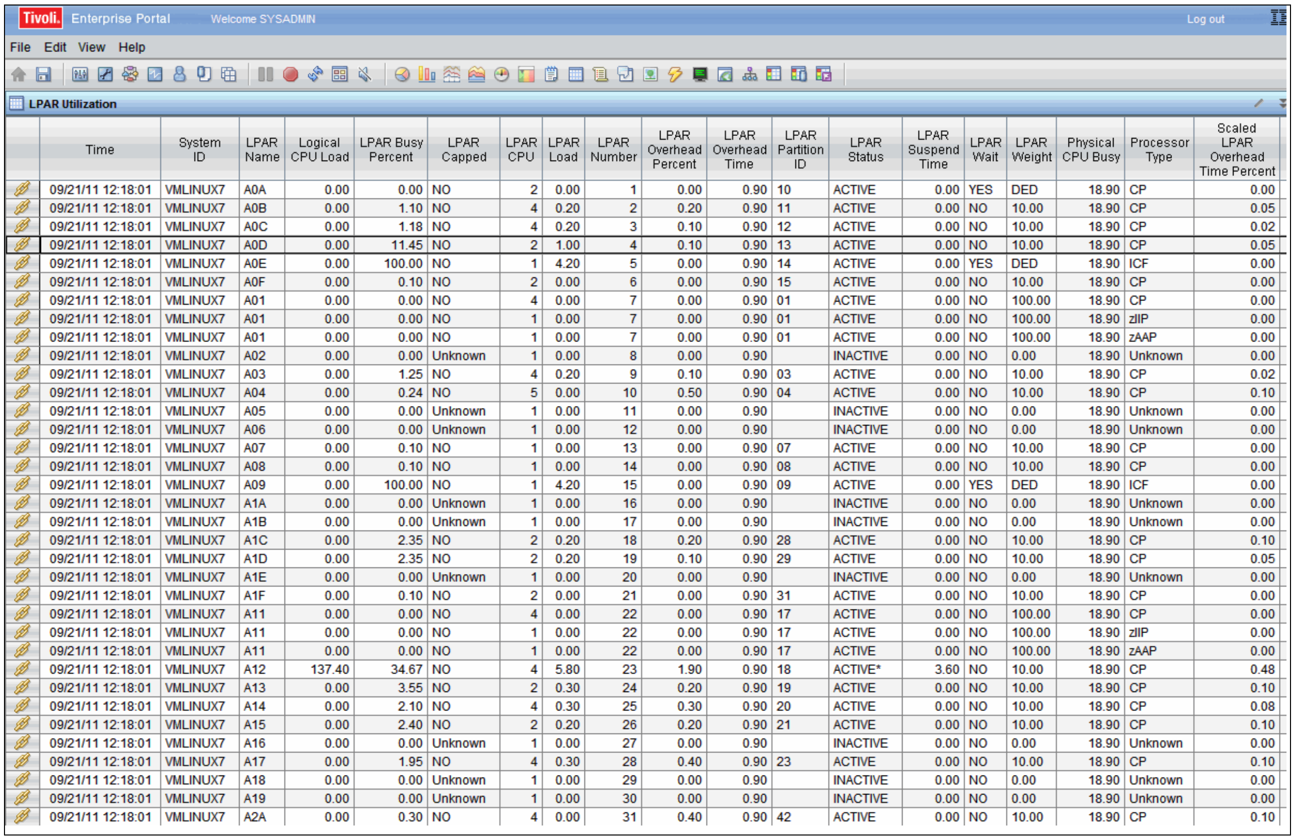
```
Interval 00:00:01-11:40:01, on 2011/09/21

Interval <Partition->                                        <- Load per Log. Processor -->
End Time Name      Nr.   Upid  #Proc Weight Wait-C Cap %Load  %Busy %Ovhd %Susp %VMld %Logld Type
11:40:01 A0F         6    15      2     10     NO  NO    .0     .1    .0   ...   ...   ... CP
11:40:01 A01         7    01      6     10     NO  NO   ...     .0    .0   ...   ...   ... MIX
11:40:01 A02         8    ..      0      0     NO  NO   ...    ...   ...   ...   ...   ... ..
11:40:01 A03         9    03      4     10     NO  NO    .2    1.5    .0   ...   ...   ... CP
11:40:01 A04        10    04      5     10     NO  NO    .0     .2    .1   ...   ...   ... CP
11:40:01 A05        11    ..      0      0     NO  NO   ...    ...   ...   ...   ...   ... ..
11:40:01 A06        12    ..      0      0     NO  NO   ...    ...   ...   ...   ...   ... ..
11:40:01 A07        13    07      1     10     NO  NO    .0     .1    .0   ...   ...   ... CP
11:40:01 A08        14    08      1     10     NO  NO    .0     .1    .0   ...   ...   ... CP
11:40:01 A09        15    09      1    DED    YES  NO   ...  100.0    .0   ...   ...   ... ICF
11:40:01 A1A        16    ..      0      0     NO  NO   ...    ...   ...   ...   ...   ... ..
11:40:01 A1B        17    ..      0      0     NO  NO   ...    ...   ...   ...   ...   ... ..
11:40:01 A1C        18    28      2     10     NO  NO    .2    2.6    .1   ...   ...   ... CP
11:40:01 A1D        19    29      2     10     NO  NO    .2    2.7    .1   ...   ...   ... CP
11:40:01 A1E        20    ..      0      0     NO  NO   ...    ...   ...   ...   ...   ... ..
11:40:01 A1F        21    31      2     10     NO  NO    .0     .1    .0   ...   ...   ... CP
11:40:01 A11        22    17      6     10     NO  NO   ...     .0    .0   ...   ...   ... MIX
11:40:01 A12        23    18      4     10     NO  NO   5.7   34.3    .4    .9  33.8  34.1 CP
11:40:01 A13        24    19      2     10     NO  NO    .4    5.2    .1   ...   ...   ... CP
11:40:01 A14        25    20      4     10     NO  NO    .4    2.1    .1   ...   ...   ... CP
11:40:01 A15        26    21      2     10     NO  NO    .2    2.8    .1   ...   ...   ... CP
```

Figure 8-6 FXC202 LPAR Partition Activity Log (LPARLOG)

If the partition does not mix engine types, the displayed averages of %Busy and %Ovhd indicate the individual partition's behavior. This environment tends to spread work out more evenly among the processors. If the partition mixes engines, the %Busy and %Ovhd still report the averages. However, the individual utilizations of the partitions might vary wildly from one processor type to another. Using the FCX126 LPAR Activity report might be the better choice. The mixed engine configuration indicator is when the Type field shows MIX. In a non-mixed configuration, the report shows the type of engine, for example, CP, IFL, ICS, ZIIP, or ZAAP.

There is a correlation between the number of logical processors defined and active, and the overhead involved in time slicing between the physical and logical processors. The more logical processors, the higher the overhead.

A third report shows another view of the same data gathered using the Tivoli Omegamon TEPS monitor (Figure 8-7). This is a summarized view of each partition running on the LPAR. We see the same information displayed in the VM Performance Toolkit reports if we look at A12.



Time	System ID	LPAR Name	Logical CPU Load	LPAR Busy Percent	LPAR Capped	LPAR CPU	LPAR Load	LPAR Number	LPAR Overhead Percent	LPAR Overhead Time	LPAR Partition ID	LPAR Status	LPAR Suspend Time	LPAR Wait	LPAR Weight	Physical CPU Busy	Processor Type	Scaled LPAR Overhead Time Percent
09/21/11 12:18:01	VMLINUX7	A0A	0.00	0.00	NO	2	0.00	1	0.00	0.90	10	ACTIVE	0.00	YES	DED	18.90	CP	0.00
09/21/11 12:18:01	VMLINUX7	A0B	0.00	1.10	NO	4	0.20	2	0.20	0.90	11	ACTIVE	0.00	NO	10.00	18.90	CP	0.05
09/21/11 12:18:01	VMLINUX7	A0C	0.00	1.18	NO	4	0.20	3	0.10	0.90	12	ACTIVE	0.00	NO	10.00	18.90	CP	0.02
09/21/11 12:18:01	VMLINUX7	A0D	0.00	11.45	NO	2	1.00	4	0.10	0.90	13	ACTIVE	0.00	NO	10.00	18.90	CP	0.05
09/21/11 12:18:01	VMLINUX7	A0E	0.00	100.00	NO	1	4.20	5	0.00	0.90	14	ACTIVE	0.00	YES	DED	18.90	ICF	0.00
09/21/11 12:18:01	VMLINUX7	A0F	0.00	0.10	NO	2	0.00	6	0.00	0.90	15	ACTIVE	0.00	NO	10.00	18.90	CP	0.00
09/21/11 12:18:01	VMLINUX7	A01	0.00	0.00	NO	4	0.00	7	0.00	0.90	01	ACTIVE	0.00	NO	100.00	18.90	CP	0.00
09/21/11 12:18:01	VMLINUX7	A01	0.00	0.00	NO	1	0.00	7	0.00	0.90	01	ACTIVE	0.00	NO	100.00	18.90	zIIP	0.00
09/21/11 12:18:01	VMLINUX7	A01	0.00	0.00	NO	1	0.00	7	0.00	0.90	01	ACTIVE	0.00	NO	100.00	18.90	zAAP	0.00
09/21/11 12:18:01	VMLINUX7	A02	0.00	0.00	Unknown	1	0.00	8	0.00	0.90		INACTIVE	0.00	NO	0.00	18.90	Unknown	0.00
09/21/11 12:18:01	VMLINUX7	A03	0.00	1.25	NO	4	0.20	9	0.10	0.90	03	ACTIVE	0.00	NO	10.00	18.90	CP	0.02
09/21/11 12:18:01	VMLINUX7	A04	0.00	0.24	NO	5	0.00	10	0.50	0.90	04	ACTIVE	0.00	NO	10.00	18.90	CP	0.10
09/21/11 12:18:01	VMLINUX7	A05	0.00	0.00	Unknown	1	0.00	11	0.00	0.90		INACTIVE	0.00	NO	0.00	18.90	Unknown	0.00
09/21/11 12:18:01	VMLINUX7	A06	0.00	0.00	Unknown	1	0.00	12	0.00	0.90		INACTIVE	0.00	NO	0.00	18.90	Unknown	0.00
09/21/11 12:18:01	VMLINUX7	A07	0.00	0.10	NO	1	0.00	13	0.00	0.90	07	ACTIVE	0.00	NO	10.00	18.90	CP	0.00
09/21/11 12:18:01	VMLINUX7	A08	0.00	0.10	NO	1	0.00	14	0.00	0.90	08	ACTIVE	0.00	NO	10.00	18.90	CP	0.00
09/21/11 12:18:01	VMLINUX7	A09	0.00	100.00	NO	1	4.20	15	0.00	0.90	09	ACTIVE	0.00	YES	DED	18.90	ICF	0.00
09/21/11 12:18:01	VMLINUX7	A1A	0.00	0.00	Unknown	1	0.00	16	0.00	0.90		INACTIVE	0.00	NO	0.00	18.90	Unknown	0.00
09/21/11 12:18:01	VMLINUX7	A1B	0.00	0.00	Unknown	1	0.00	17	0.00	0.90		INACTIVE	0.00	NO	0.00	18.90	Unknown	0.00
09/21/11 12:18:01	VMLINUX7	A1C	0.00	2.35	NO	2	0.20	18	0.20	0.90	28	ACTIVE	0.00	NO	10.00	18.90	CP	0.10
09/21/11 12:18:01	VMLINUX7	A1D	0.00	2.35	NO	2	0.20	19	0.10	0.90	29	ACTIVE	0.00	NO	10.00	18.90	CP	0.05
09/21/11 12:18:01	VMLINUX7	A1E	0.00	0.00	Unknown	1	0.00	20	0.00	0.90		INACTIVE	0.00	NO	0.00	18.90	Unknown	0.00
09/21/11 12:18:01	VMLINUX7	A1F	0.00	0.10	NO	2	0.00	21	0.00	0.90	31	ACTIVE	0.00	NO	10.00	18.90	CP	0.00
09/21/11 12:18:01	VMLINUX7	A11	0.00	0.00	NO	4	0.00	22	0.00	0.90	17	ACTIVE	0.00	NO	100.00	18.90	CP	0.00
09/21/11 12:18:01	VMLINUX7	A11	0.00	0.00	NO	1	0.00	22	0.00	0.90	17	ACTIVE	0.00	NO	100.00	18.90	zIIP	0.00
09/21/11 12:18:01	VMLINUX7	A11	0.00	0.00	NO	1	0.00	22	0.00	0.90	17	ACTIVE	0.00	NO	100.00	18.90	zAAP	0.00
09/21/11 12:18:01	VMLINUX7	A12	137.40	34.67	NO	4	5.80	23	1.90	0.90	18	ACTIVE*	3.60	NO	10.00	18.90	CP	0.48
09/21/11 12:18:01	VMLINUX7	A13	0.00	3.55	NO	2	0.30	24	0.20	0.90	19	ACTIVE	0.00	NO	10.00	18.90	CP	0.10
09/21/11 12:18:01	VMLINUX7	A14	0.00	2.10	NO	4	0.30	25	0.30	0.90	20	ACTIVE	0.00	NO	10.00	18.90	CP	0.08
09/21/11 12:18:01	VMLINUX7	A15	0.00	2.40	NO	2	0.20	26	0.20	0.90	21	ACTIVE	0.00	NO	10.00	18.90	CP	0.10
09/21/11 12:18:01	VMLINUX7	A16	0.00	0.00	Unknown	1	0.00	27	0.00	0.90		INACTIVE	0.00	NO	0.00	18.90	Unknown	0.00
09/21/11 12:18:01	VMLINUX7	A17	0.00	1.95	NO	4	0.30	28	0.40	0.90	23	ACTIVE	0.00	NO	10.00	18.90	CP	0.10
09/21/11 12:18:01	VMLINUX7	A18	0.00	0.00	Unknown	1	0.00	29	0.00	0.90		INACTIVE	0.00	NO	0.00	18.90	Unknown	0.00
09/21/11 12:18:01	VMLINUX7	A19	0.00	0.00	Unknown	1	0.00	30	0.00	0.90		INACTIVE	0.00	NO	0.00	18.90	Unknown	0.00
09/21/11 12:18:01	VMLINUX7	A2A	0.00	0.30	NO	4	0.00	31	0.40	0.90	42	ACTIVE	0.00	NO	10.00	18.90	CP	0.10

Figure 8-7 Tivoli Omegamon TEPS Monitor - LPAR View

8.1.2 LPAR overhead

In VM Performance Toolkit, the LPAR overhead for managing partitions is reported in two values. Each partition is associated with an overhead (%Ovhd) value. For example, in Figure 8-5 on page 131, you can view the overhead for each engine. The %Ovhd for A12 is .4%. Overhead is calculated as the percent of elapsed time that the LPAR has spent running and controlling the partitions. In Figure 8-5 on page 131, the total overhead for all processors in all partitions is .9%.

Note: The IBM VM Performance Toolkit report shows all processor types:

- General use central processors (CPs)
- Integrated Facility for Linux (IFL)
- Internal Coupling Facility (ICF)
- System z noIntegrated Information Processor (zIIP)
- System z Application Assist Processors (zAPP)

Forty-five partitions are defined in the report. There are 88 logical processors shared between active guests, with 20 physical (that is, non-dedicated) processors to which they can be dispatched:

- Ten CP engines
- Two ZAAP engines
- Two ZIIP engines
- Two IPL engines
- Four ICF engines

Each has an overhead of 2 - 4%. The dedicated processors have no associated overhead in the example. However, we do not suggest that you dedicate processors to LPARs to reduce overhead because doing so makes dedicated processors unavailable to other LPARs in the configuration. The number of LPARS defined, plus the number of logical processors defined clearly relates to the total overhead costs. You can see that some time will always be associated with overhead, whether it is associated with the partition itself or with the operating system. To reduce physical overhead, try using fewer LPARs and fewer logical processors.

8.1.3 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 and engine type. Use the following calculation to determine the weight of each logical processor:

1. Add all the weights for the logical processors.

   In the previous example, there are three dedicated partitions and 45 sharing LPARs defined, 32 of which are in use, sharing 20 logical processors based on weighting. The total weight of all the logical processors is 280.

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 (10/280) yields a 3% share of the 20 shared processors.

3. Divide the number of LPAR logical processors 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 operates if capped. Thus, 3% of 20 processors is equivalent to almost half of one processor (3 x 20 = 60%). Divide this between the four logical processors used by our VM system in LPAR A12, and each processor is allocated 15% of one processor.

> **Note:** This calculation always applies, even when the LPAR runs at less than 100% capacity. If an LPAR does not use its allocation, the available 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 time slicing, 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 have any activity. Thus, the A12 LPAR could get up to 90% of each logical engine in its assigned time, with all of the total available free capacity that is not being used or dedicated to other LPARS.

8.1.4 LPAR analysis example

If the weight of an LPAR is 10, and the total weight 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 configuration decreases the relative speed of each logical processor in an LPAR.

Note: Use a standard base such as 100 or 1000 as the weight to set weights for LPARs. For example, if you have four LPARs, one weighted at 5000 and the other three at 100, the total of all LPARs is 5300. The largest LPAR has a relative weight of 94% (5000 / 5300 = 94%). If you use 1000, and the largest LPAR has a weight of 940 and the other three LPARs have a weight of 20 each, the largest LPAR still has the relative weight of 94% (940 / 1000=94%). Therefore, using 100 or 1000 makes it easier for you to see relative weights of an LPAR at a glance.

8.1.5 LPAR options

You can define LPAR shares as capped, indicating that their share of the physical system is capped to their given share. There would be a small amount of resources if 1% was allocated. But if you define the LPAR as not capped, the unused processor resources are available to any LPAR that can use the resources, based on given weights. Use capped shares only in exceptional circumstances, for example, in installations where resources need to be strictly limited, or when a financial agreement exists to provide a specific speed of processor. In all other instances, we recommend that you run uncapped, and allow PR/SM™ to allocate processor resources as required across active LPARs.

You set time slicing to either specific or dynamic. Specific time slices are rarely used because they can cause erratic responsiveness from the processor subsystem. Wait completion defines whether LPARs give up the processor when there is no work, or keep it for the remaining time slice. When you enable wait completion, the LPAR gives the processor when work is complete.

8.1.6 Capacity planning view

Capacity planning is the long range view of the environment. This planning uses current system information to determine when you need additional hardware to handle workloads. Capacity planning not only analyzes peaks and the averages, but determines what is driving the workload and plans for additional work.

If you are using many LPARs on a VM system, you need to know how fully loaded the system is when all partitions are active. An effective way to do this is to create a chart using cumulative stacking of processor utilization of all partitions over time. For capacity planning, at least six months of data is beneficial because it shows trends along with peaks and valleys, and potential seasonality of the data. You can extract performance data from z/VM and load it into a spreadsheet or graphics package.

You can see CPU utilization of the other VM, VM/VSE, and z/OS LPARS, in addition to others running VM with Linux guests, all from a VM system running in LPAR mode. Figure 8-8 shows the utilization trend over a six-month period. You can also add a trend line to project when the current hardware might be out of capacity. In the current view, you see that there is enough capacity for the current environment for at least the next several months. You might want to add additional IFL to this environment for future growth. You can also determine which workloads are the main contributors when the system is reaching an overall critical processor load state.

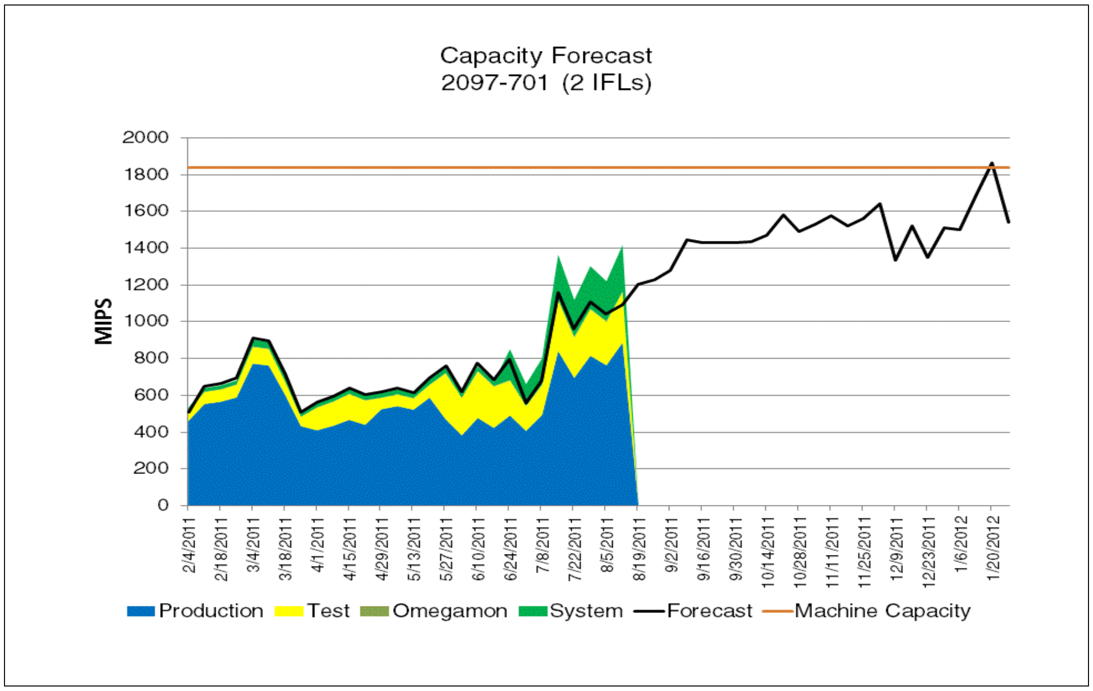


Figure 8-8 Example of cumulative stacked chart for capacity planning

8.1.7 Shared versus dedicated processors

You can dedicate processors to an LPAR when multiple physical processors are being used by many different logical partitions. Generally, you choose this option for two reasons:

- As benchmarks to reduce questionable impacts from other workloads. Be sure to always factor out interference from other work running in the same time frame as completely as possible. Dedicating processors means that the processor availability is constant at all times.
- When the workload is steady enough to utilize the processors, and there are sufficient resources to justify dedicated processors. A good example is a mission-critical partition that runs at consistently high processor utilization.

Important: For consistency reasons, always use dedicated processors to reduce the impact of other workloads on the results when running benchmarks.

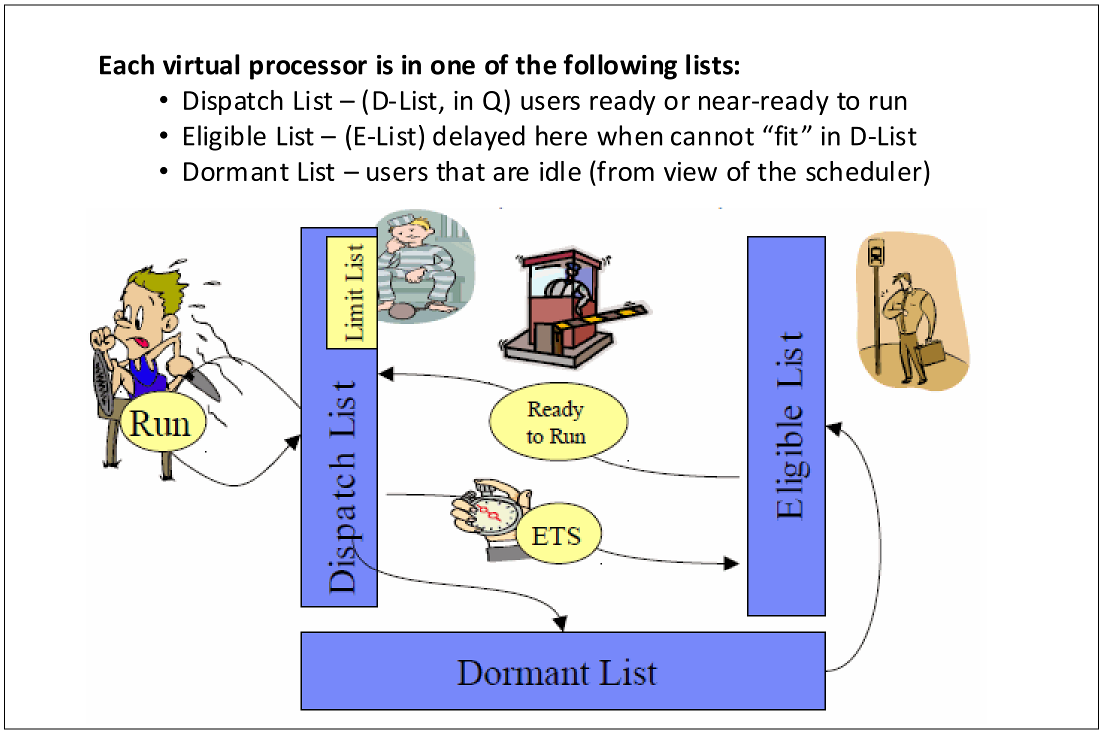
Your objective should always be high utilization in order to justify the cost of System z implementations. Utilizations of over 90%, approaching 100%, are common in System z, and sustainable in workload throughput and response time. High utilization leverages the value of the reliability, availability, and serviceability of the System z systems. Other platforms rarely operate at high utilizations, many midrange systems peaking at 30%, and smaller server platforms at even lower levels. You risk reducing the overall system utilization when you dedicate physical resources (for example, processors) to one LPAR. This can also reduce System z effectiveness and increase the cost per unit of work.

8.2 z/VM 6.1 and the CP Scheduler

The CP Scheduler uses time-sharing principles to keep as many of the logged-on virtual machines as possible operating concurrently. It contains algorithms that consider factors such as the availability of processing time, paging resources, and real storage, as compared with virtual machine requirements. The CP Scheduler uses two time slices to determine how long a given virtual machine competes for access to the processor:

- Elapsed time slice where virtual machines compete for processor use for the duration of the elapsed time slice
- Dispatch time slice where a virtual machine can only control the processor for a duration of its dispatch time slice, also referred to as minor time slice

When the dispatch time slice for a virtual machine expires, CP Scheduler readjusts its priority relative to other virtual machines competing for the processor. When its elapsed time slice expires, CP Scheduler drops the virtual machine from the set competing for the processor. CP Scheduler then tries to add another virtual machine eligible for processor resources into the competing set. Figure 8-9 shows an overview of this process.



Each virtual processor is in one of the following lists:

- Dispatch List – (D-List, in Q) users ready or near-ready to run
- Eligible List – (E-List) delayed here when cannot “fit” in D-List
- Dormant List – users that are idle (from view of the scheduler)

Figure 8-9 CP Scheduler scheme

8.2.1 Transaction classification

Virtual machines are classified according to their transaction characteristics and resource requirements for purposes of dispatching and scheduling:

- Class 1: Virtual machines designated as interactive tasks.
- Class 2: Virtual machines designated as non-interactive tasks.
- Class 3: Virtual machines designated as resource-intensive tasks.
- Class 0: Special class designation for virtual machines that require immediate access to processor resources. This class needs a special privilege called quick dispatch.

Started virtual machines reside on one of three lists for processor scheduling:

- The dormant list
- The eligible list
- The dispatch list

8.2.2 The dormant list

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

8.2.3 The eligible list

The eligible list consists of virtual machines not currently being considered for dispatching due a system resource constraint, for example, paging or storage. Virtual machines are kept back 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 workload requirements:

- E1: Class 1 virtual machines that are expected to perform short transactions. Virtual machines are classified as E1 when they enter the eligible list for the first time.
- E2: Class 2 virtual machines that are expected to perform medium-length transactions. These virtual machines drop to the eligible list after spending at least one elapsed time slice in E1 without completing processing.
- E3: Class 3 virtual machines that are 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 never 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 8.4.3, “The CP QUICKDSP option” on page 146.

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 by the Scheduler when they move to the dispatch list. Priorities assigned in this way are intended to:

- Slow down virtual machines that are resource intensive in favor of virtual machines requiring fewer resources.
- Ensure that virtual machines receive a designated portion of the processor. See 8.5, “CP set share command” on page 147, for details.
- Control the amount and type of service based on virtual machine classification:
  - E1
  - E2
  - E3

We discuss how the Scheduler calculates priorities in 8.2.4, “The dispatch list” on page 139.

8.2.4 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. These are directly analogous to the E1, E2, E3, and E0 classifications while on the eligible list.

Note: EO virtual machines on the eligible list are included in the count of Q0 virtual machines displayed by the `cp indicate load` command.

8.3 Virtual machine scheduling

VM definition blocks represent virtual processors allocated to a virtual machine. By default, every virtual machine has at least one definition block, or the base. Definition blocks are discussed in 8.3.4, “Scheduling virtual processors” on page 141. Figure 8-10 shows VM definition blocks on the dispatch and eligible lists and illustrates the state transitions involved in scheduling a virtual machine.

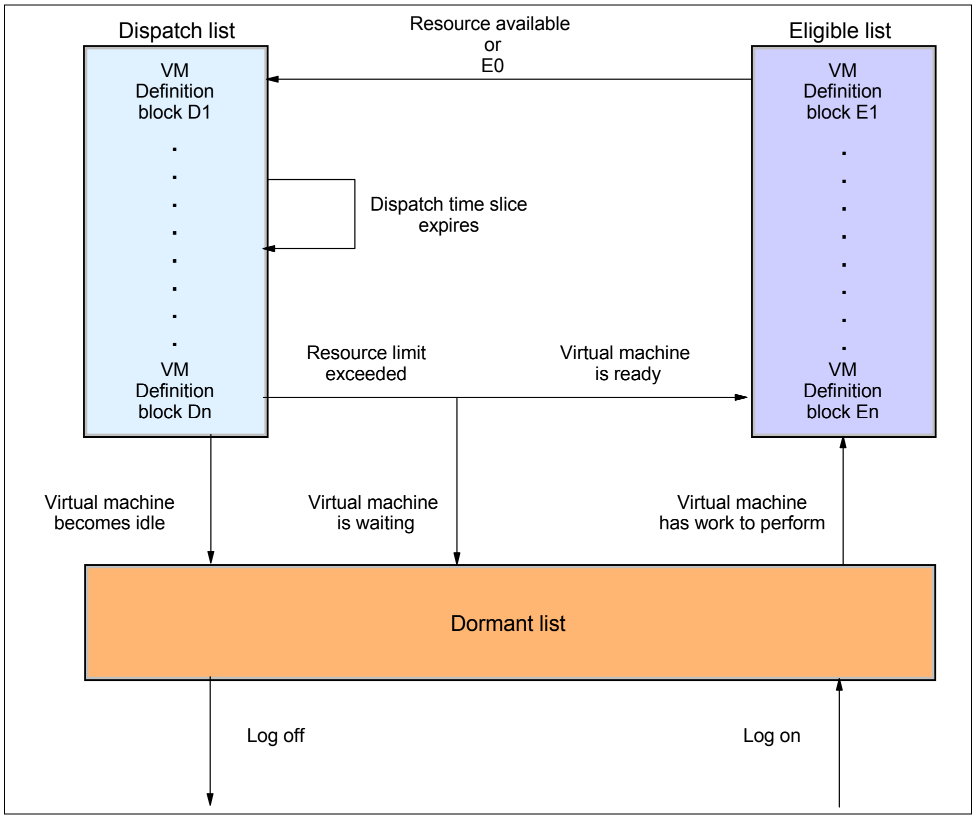


Figure 8-10 CP scheduling: State transitions

8.3.1 Dormant list

Virtual machines are placed on the dormant list on logon. They enter the dormant list from the dispatch list when they complete a transaction. This is known as being dropped from the queue. A virtual machine can also enter the dormant list from the dispatch list if its elapsed time slice has expired and it is waiting for a resource, such as a demand page-in. Virtual machines are moved to the eligible list when processor time is required.

8.3.2 Eligible list

Virtual machines are classified as E1 when they enter the eligible list from the dormant list. 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, the E2 drops to E3, and E3 remains in E3.

8.3.3 Dispatch list

A virtual machine must pass these three tests to move from the eligible list to the dispatch list:

- Storage test

  The virtual machine associated working set must fit in real storage as it stands at that time. See “STORBUF control” on page 143 for details.
- Paging test

  If the user is a loading user, there must be room in the loading user buffer. Loading users are discussed in “LDUBUF control” on page 144.
- Processor test

  There must be room in the dispatch buffer, as discussed in “DSPBUF control” on page 143.

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

During initialization, the CP Scheduler assigns an initial elapsed time slice value of 1.2 seconds for E1 virtual machines. 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 value will always be between 50 ms and 16 seconds. E0, E2, and E3 elapsed time slices are assigned values of 6, 8, and 48 times larger, respectively, than the E1 time slice.

The dispatch time slice is computed at CP initialization. You can use the **`query srm dspslice`** to query the dispatch time slice value.

A virtual machine runs for its designated elapsed time slice while on the dispatch list. If it does not complete processing in that period, it is moved to back to the eligible queue with a lower workload classification, if not already in E3. If the resource is capped and has used all of its available slice, it also moves back to the eligible queue.

8.3.4 Scheduling virtual processors

Additional VM definition blocks are created for each virtual processor defined to a virtual machine. These additional blocks 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. The list hierarchy is, from highest to lowest:

- Dispatch
- Eligible
- Dormant

This ensures that virtual machine resource requirements 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 in scheduling that block on the dispatch list.

To illustrate, consider a virtual machine with two virtual processors, both starting in the dormant list, along with two VM corresponding definition blocks. If the additional definition block becomes dispatchable, both definition blocks, base and additional, move to the eligible list. After waiting in the eligible list, both definition blocks again move to the dispatch list with the same priority. As the additional definition block consumes resources, its intermediate dispatch priority is calculated independently from the base block definition. We discuss virtual processors in 9.3, “Performance effect of virtual processors” on page 160.

8.3.5 z/VM scheduling and the Linux timer patch

The Linux kernel tracks time with a timer that interrupts at a constant rate. The kernel maintains a regularly updated global variable, jiffies, and inspects queues for work waiting to be done. In a Linux guest on z/VM, these tasks interfere with the z/VM Scheduler. CP classified a Linux virtual machine as a long-running task and assigns it to Q3 (while using 10 ms timer interrupt). This prevents z/VM from distinguishing between running and idling guests.

You can switch the timer interrupts on and off on demand. With the timer on, Linux virtual machines are dispatched less frequently when they are idling. This significantly reduces 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.

8.4 CP scheduler controls

You can influence the CP Scheduler with these controls:

- SRM controls globally influence the overall processor resources.
- Local controls influence how is the Scheduler sees an individual virtual machine.

8.4.1 When to use SRM controls

Use SRM controls for these:

- Make sure that processor resources are utilized as much as possible.
- Thrashing does not occur.
- Particular servers get preferential access to processor resources when performance is down.

8.4.2 Global System Resource Manager (SRM) controls

Use the **`cp set srm`** command to tune z/VM processor resources. The original default values were determined based on a traditional workload mix of interactive CMS and guest work. We advise that you change only one SRM value at a time and examine the results before you change more controls.

DSPBUF control

Use the **`cp set srm dspbuf`** command to control the absolute number of virtual processors allowed in the dispatch list for each scheduling class. This command allows you to overcommit or undercommit processor and I/O device resources. This is the command format:

```
set srm dspbuf n1 n2 n3
```

Where:

n1	Number of class 1 virtual processors permitted in the dispatch list.
n2	Number of class 2 virtual processors permitted in the dispatch list.
n3	Number of class 3 virtual processors permitted in he dispatch list.

The default value is 32767 32676 32676, which basically indicates OFF. We recommend that you do not adjust this control. Rather, allow CP algorithms to use dynamic control, dependent on the changing system conditions.

STORBUF control

Use the **`cp set srm storfub`** command to partition real storage commitment in terms of pages based on the user's transaction class (that is, E1, E2, or E3). This allows you to overcommit or undercommit real storage and protects the real memory from general thrashing. We suggest that you examine this SRM value first whenever your system is not performing well.

This is the command format:

```
set srm storbuf p1 p2 p3
```

Where:

p1	Maximum percentage of system storage available to E1, E2, and E3 users.
p2	Maximum percentage of sstem storage available to E2 and E3 users.
p3	Maximum percentage of system storage available to E3 users.

Valid operand ranges are $999 \geq i \geq j \geq k \geq 0$. STORBUF default values are 125, 105, and 95.

You can gain performance gains by overcommitting the overall system real storage when expanded storage is available. When you overcommit, a virtual machine's working set is computed as before, but the apparent real storage available to virtual machines waiting to enter the dispatch list appears larger. Therefore, more virtual machines are allowed into the dispatch list. You might see higher paging rates, but the benefit of reducing the eligible list and moving users into the dispatch list offsets this increase. Expanded storage acts as a very effective, fast page buffer to real storage.

CP Scheduler sees the storage requirement for virtual machines that do not drop from queue to be larger than actual. Because the Scheduler cannot determine the real storage requirements, raising the storbuf control can improve system performance. Note that this effectively disables the storage test discussed in 8.2.4, "The dispatch list" on page 139.

Setting the storfub control for Linux guests has not been precise, resulting in, for example, little difference between a storbuf of 300 versus one of 900. The storage test in both cases was probably disabled. However, in a large CMS group of users, storbuf can be an effective control mechanism. You can experiment with overcommitting resources with storbuf, given the recent improvements in z/Linux in guest mode, particularly timer modifications.

LDUBUF control

The **`cp set srm ldubuf`** command partitions the commitment of the system paging resources and protects the system from thrashing DASD paging. Thrashing occurs in paging when contention for paging and storage devices results in less work being accomplished. You can use **`cp set srm ldubuf`**, especially if using **`storbuf`** has not given clear results, whenever you are dealing with performance issues.

LDUBUF is a loading user buffer. A loading user is a heavy user of paging resources with an expected high paging rate and is defined as a user who takes five page faults in one time slice of work. Because a time slice is relatively small, a user taking five page faults in that time is consuming the equivalent of one paging device. You can use the **`indicate queues exp`** command to display current loading users.

This is the command format:

```
set srm ldubuf p1 p2 p3
```

Where:

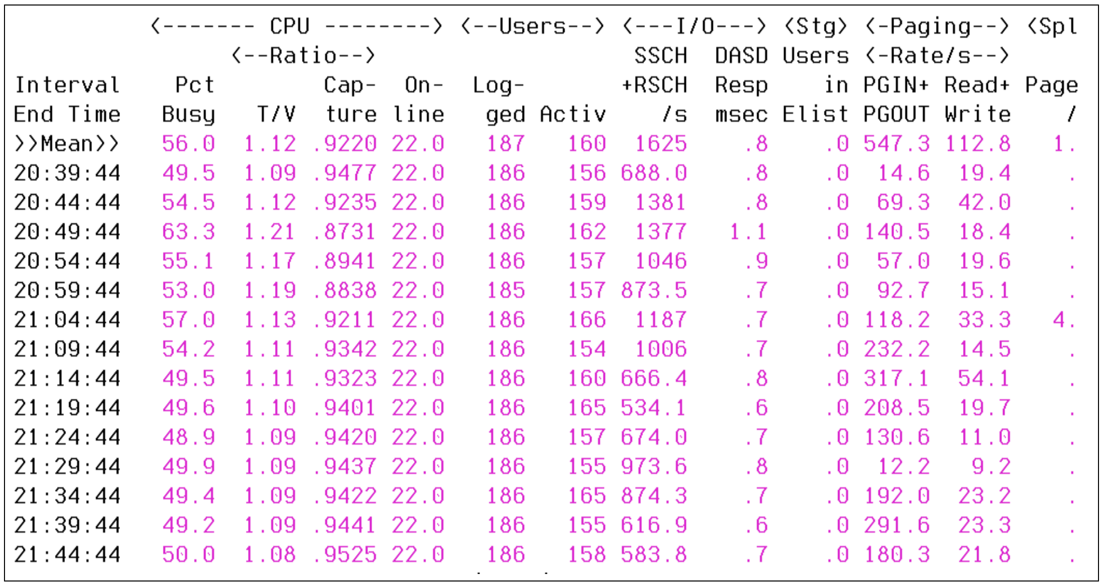
p1	Percentage of system paging resources available to E1, E2, and E3 users.
p2	Percentage of system paging resources available to E2 and E3 users.
p3	Percentage of system paging resources available to the E3 user.

ldubuf default values are 100, 85, and 65.

Here we discuss key points of ldubuf. When the Schedule evaluated loading capacity, the number of paging devices is the loading capacity. When you set ldubuf to the default values, 100% of the total capacity, or the number of paging devices, can be utilized. For example, with seven paging devices, seven users considered as loading are allowed on the dispatch list. Additional users are kept on the eligible list until an existing loading user either drops from the list or obtains its working set in storage and stops causing page faults.

We used a benchmark typical of Linux environments, that is, one that allocated storage, performed a function, and released the storage. Measurements show that ldubuf (at the default level) allows the total paging subsystem to be 100% consumed.

Figure 8-11 shows CPU utilization and paging rates for ldubuf and storbuf in default settings. This report displays important performance metrics, with the columns PGIN+POUT and Read+Write showing system basic paging rates. PGIN+PGOUT shows the paging rate to or from XSTORE. Read+Write shows the paging rate to or from DASD. The report shows 547.3 pages/second in XSTORE, and the paging activity to or from DASD is 112.8 pages/second. If the system is paging more to DASD than to XSTORE, you might need to increase XSTORE size.



```
         <------- CPU --------> <--Users--> <---I/O---> <Stg> <-Paging--> <Spl
               <--Ratio-->                        SSCH  DASD Users <-Rate/s-->
Interval   Pct        Cap-  On-  Log-            +RSCH  Resp    in PGIN+ Read+ Page
End Time   Busy   T/V  ture line  ged Activ        /s  msec Elist PGOUT Write    /
>>Mean>>   56.0  1.12 .9220 22.0  187   160  1625    .8    .0 547.3 112.8   1.
20:39:44   49.5  1.09 .9477 22.0  186   156 688.0    .8    .0  14.6  19.4    .
20:44:44   54.5  1.12 .9235 22.0  186   159  1381    .8    .0  69.3  42.0    .
20:49:44   63.3  1.21 .8731 22.0  186   162  1377   1.1    .0 140.5  18.4    .
20:54:44   55.1  1.17 .8941 22.0  186   157  1046    .9    .0  57.0  19.6    .
20:59:44   53.0  1.19 .8838 22.0  185   157 873.5    .7    .0  92.7  15.1    .
21:04:44   57.0  1.13 .9211 22.0  186   166  1187    .7    .0 118.2  33.3   4.
21:09:44   54.2  1.11 .9342 22.0  186   154  1006    .7    .0 232.2  14.5    .
21:14:44   49.5  1.11 .9323 22.0  186   160 666.4    .8    .0 317.1  54.1    .
21:19:44   49.6  1.10 .9401 22.0  186   165 534.1    .6    .0 208.5  19.7    .
21:24:44   48.9  1.09 .9420 22.0  186   157 674.0    .7    .0 130.6  11.0    .
21:29:44   49.9  1.09 .9437 22.0  186   155 973.6    .8    .0  12.2   9.2    .
21:34:44   49.4  1.09 .9422 22.0  186   165 874.3    .7    .0 192.0  23.2    .
21:39:44   49.2  1.09 .9441 22.0  186   155 616.9    .6    .0 291.6  23.3    .
21:44:44   50.0  1.08 .9525 22.0  186   158 583.8    .7    .0 180.3  21.8    .
```

Figure 8-11 FCX225 CPU and paging rates (SYSSUMLOG)

MAXWSS control

Use the **cp set srm maxwss** command to set the maximum working set allowed for a normal user on the dispatch list. The user is moved back to the eligible list whenever the working set size exceeds this percentage. This is the command format:

```
set srm maxwss m
```

m specifies the maximum percentage of system pageable storage available to a user. maxwss prevents a large virtual machine from acquiring an excessive amount of system memory at the expense of smaller users. For maxwss to work effectively, virtual machines must reside in the eligible list. That is, the Scheduler must consider the system to be storage constrained based on storbuf settings.

DSPSLICE control

A virtual machine is assigned a dispatch time slice each time that it is dispatched. The dispatch time slice represents a fixed number of instructions processed, and is calculated at initialization, based on real processor speed. You can change this value with the **cp set srm dspslice** command in this format:

```
set srm dspslice min
```

min specifies the minimum dispatching time slice in milliseconds. Valid dispatching time slice values are in the range $100 \geq min \geq 1$. Currently, dispatching overhead might not justify increasing this value.

After it is dispatched, a virtual machine runs until:

- Its minor time slice interval expires.
- It completes the workload and is waiting for more.
- It enters CP mode.
- An interrupt occurs.

A new dispatch time slice is also assigned when a machine is redispatched. If it relinquished the processor due to an interrupt, it is assigned the remaining portion of the previous dispatch time slice. For example, even when performance is low, you still need TCP/IP, or a DNS server or security manager, to perform well. You need to use SRM controls properly and not overuse quickdsp. See 8.4.3, “The CP QUICKDSP option” on page 146, for details. We suggest that you use quickdsp for machines that need to run well even when performance is low, for example, mission-critical processes and workloads.

Use storbuf to over-allocate storage if virtual machines do not drop from the queue, usually because dispatching and scheduling algorithms are skewed. The timer modification must be installed for Linux guests to drop from the queue. See 8.3.5, “z/VM scheduling and the Linux timer patch” on page 142, for details. Make sure to use an appropriate network architecture, as discussed in 11.4.1, “Qeth device driver for OSA-Express” on page 204.

It is difficult to know storage requirements and even more difficult to perform scheduling when the machines do not drop from the queue. We found that the perceived storage requirement is very high in installations running many servers, even idle servers that never drop from the queue. You can use high storbuf values to over-allocate storage and improve performance. However, it might not be effective to use storbuf in situations where virtual machines do drop from the queue. Instead, set the **`set srm xstore`** command to a minimum of 50%, and up to a maximum value, provided that you have expanded storage.

If your environment performs considerable paging, control it with the **`set srm ldubuf`** command. The default ldubuf allows the paging subsystem to be 100% consumed, as discussed in 8.6, “ATOD deadline versus consumption” on page 150. When the paging subsystem is at 100% utilization, there is not much advantage to increasing ldubuf and allowing more users to consume paging resources.

8.4.3 The CP QUICKDSP option

Use the **`set quickdsp`** command for service machines that must run even during serious memory or processor constraints. These machines are usually classified as E0, which means that they are added immediately to the dispatch list whenever work is required without waiting in the eligible list. OFF is the default for **`quickdsp`**. Example 8-1 shows the **`set quickdsp`** syntax.

Example 8-1 SET QUICKDSP command

```
set quickdsp mntlinux on
USER MNTLINUX:  QUICKDSP = ON
Ready; T=0.01/0.01 14:24:04
```

For further details about these commands, see the z/VM Help information or z/VM CP Commands and Utilities Reference, SC24-6175.

8.5 CP set share command

The share option is an important system function that prioritizes work on the system. VM uses the `share` command to establish resource levels for CMS users, Linux guests, and service virtual machines. First decide whether the share will be absolute or relative. Then, determine the amount of that share type to allocate to each user. Consider the entire group of users that reside on the system, those that appear after system creation, and those present each time that the share is changed. Your decisions will impact Dispatch List priority calculations on the user ID and the system. Shares are normalized before being used with other users in the Dispatch List and the Eligible List per processor type. This is key in understanding how this affects system performance and who gets dispatched.

A relative share allocates a server a relative share of the processor, relative to all virtual machines in the dispatch and eligible lists. The share drops as more users log on. This works well in a large community of CMS users, even though the exact priority of a user might not be determined. A relative share evolves with the changing numbers of users, making way for newly logged on users, and ensuring that the fair sharing process is maintained when users leave the system.

Absolute shares remain fixed up to the point where the sum of the absolute shares is 100% or more, and then all of the absolute shares for each user are established by using the sum of all absolute shares allocated on the system as the divisor. An absolute share of 10% equals 10/100%, until more than 100% is allocated across the system. If 120% is allocated, the 10% share is then equal to 10/120%. At that point, all absolute shares for each user are established by using the sum of all absolute shares allocated on the system as the divisor. An absolute share of 10% equals 10/100%, until more than 99% is allocated across the system. If 120% is allocated, the 10% share is then equal to 10/120%.

8.5.1 Major settings of the share option

The share option includes the following settings:

- CPU Type:
  - CP
  - IFL
  - ZAAP
  - ZIIP
  - ICF
- Target Minimum.
- Target Maximum (Limit Share).
- For Min/Max:
  - Absolute: Percent of system summarized and normalized to 99%
  - Relative: 1 to 10,000, with the larger the number being the more important
    - Relative to other virtual machines in the Dispatch List. As the system gets busier with more work in the Dispatch List, the relative share is normalized to a smaller value.
    - Normalized to 100% - sum of absolutes.

- Additional settings for Command and Directory:
  - For example, set share *userid* REL 2000 ABS 20% LIMITHARD.
  - Second value is Limit Share.
  - Limit Share can be relative or absolute.
- LIMITSOFT: Allow the virtual machine to use more than this amount only if no other virtual machine needs the resources.
- LIMITHARD: Do not give the virtual machine more than this amount even if it is available (that is, CAPPED).
- ABSOLUTE LIMITHARD is managed differently from RELATIVE base on the **`set srm limithard`** setting added by VM64721. Problems can occur when absolute shares exceeded 100%. You might also see problems when you set a guest to a low RELATIVE target minimum share and then set a larger ABSOLUTE target maximum share. A third area of possible problems is when you set limit shares on a system with relatively low overall utilization (for example, below 60%). A solution is to set the minimum and maximum to the same absolute value, or to decrease the limit share value below the target.

8.5.2 Specialty engines and share settings

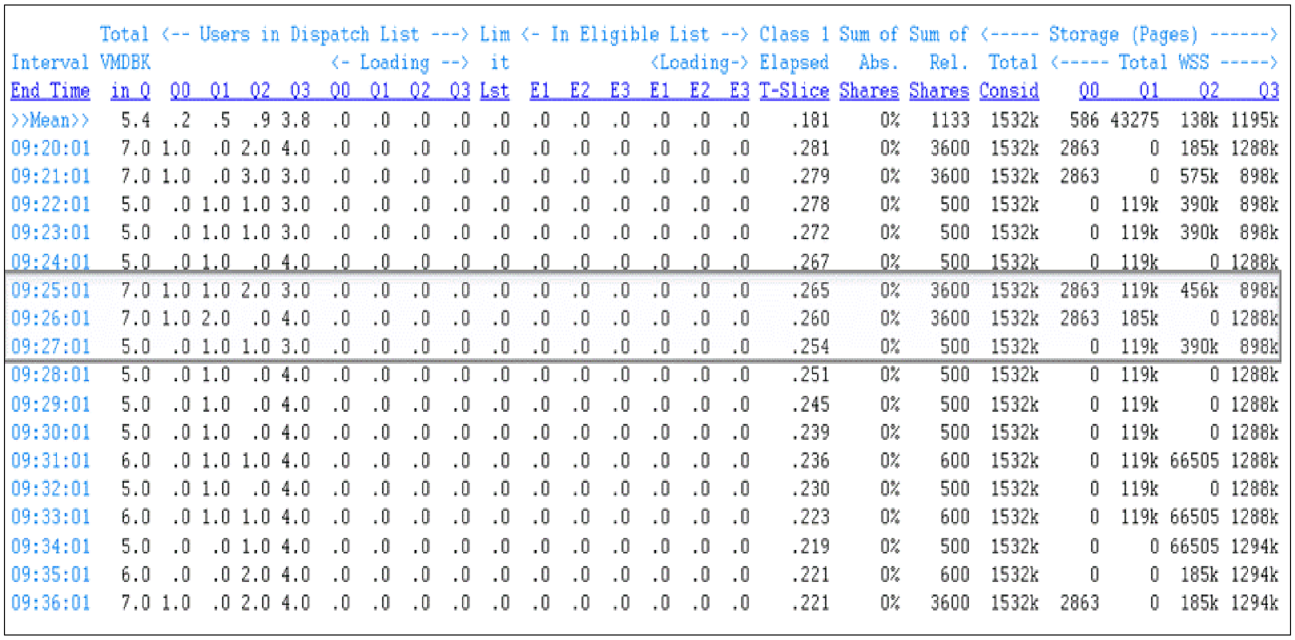
Note these concepts regarding specialty engines and share settings:

- The share setting for a virtual machine applies to each pool of processor types CP, IFL, ZAAP, ZIIP, and ICF.
- z/VM 5.4.0 added support for a separate share setting for each processor type pool. The default is TYPE ALL, which uses one setting for all types.
- Share is normalized to the sum of shares of virtual machines in the Dispatch List for each pool of processor types.
- Absolute, and normalized, is a percentage of resources of a processor type.
- Dedicated engines change what resources are divided among shares.

TCP/IP and DNS servers might have a requirement that rises as the level of work increases. Assign absolute shares to these servers. Other service machines also have load-related requirements, for example, the CP monitor that produces more records as more users log on. Consider allocating small absolute shares to **`monwrite`** and the Performance Tool Kit. Generally, you can allot a relative share to all other users. Assign a relative share higher than the default of 100 to important machines that do not need an absolute share and to important users.

VM allocates resources at run time by considering the absolute share holders first, then moving to the relative share holders. Note that all the shares relate to the currently scarce resource. Though this is generally CPU, other key resources can be constrained.

Figure 8-12 shows how a system views queues. We highlighted a section in this report to illustrate the queue length and the Relative Shares. This log file has an entry once a minute. We have one user in Q0 (quickdsp), one interactive user in Q1, two users in the Q2 Dispatch List, and three users in the Q3 Dispatch List. Note that there is active work waiting to be dispatched. There are no tasks in the Eligible List. There is .265, the length of the class 1 elapse time slice, in Class 1 Elapse T-Slice. This is the elapsed time that a class 1 user can spend in the Dispatch List and that allows 85% of all transactions to complete. This value is continuously adapted to the changing system load conditions. It can be a good response time indicator provided that you compare only periods with similar job mixes.



```
         Total <-- Users in Dispatch List ---> Lim <- In Eligible List --> Class 1 Sum of Sum of <----- Storage (Pages) ------>
Interval VMDBK                   <- Loading --> it                <Loading-> Elapsed   Abs.   Rel.  Total <----- Total WSS ----->
End Time  in Q  Q0  Q1  Q2  Q3  Q0  Q1  Q2  Q3 Lst  E1  E2  E3  E1  E2  E3 T-Slice Shares Shares Consid    Q0    Q1    Q2    Q3
>>Mean>>   5.4  .2  .5  .9 3.8  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .181     0%   1133  1532k   586 43275  138k 1195k
09:20:01   7.0 1.0  .0 2.0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .281     0%   3600  1532k  2863     0  185k 1288k
09:21:01   7.0 1.0  .0 3.0 3.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .279     0%   3600  1532k  2863     0  575k  898k
09:22:01   5.0  .0 1.0 1.0 3.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .278     0%    500  1532k     0  119k  390k  898k
09:23:01   5.0  .0 1.0 1.0 3.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .272     0%    500  1532k     0  119k  390k  898k
09:24:01   5.0  .0 1.0  .0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .267     0%    500  1532k     0  119k     0 1288k
09:25:01   7.0 1.0 1.0 2.0 3.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .265     0%   3600  1532k  2863  119k  456k  898k
09:26:01   7.0 1.0 2.0  .0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .260     0%   3600  1532k  2863  185k     0 1288k
09:27:01   5.0  .0 1.0 1.0 3.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .254     0%    500  1532k     0  119k  390k  898k
09:28:01   5.0  .0 1.0  .0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .251     0%    500  1532k     0  119k     0 1288k
09:29:01   5.0  .0 1.0  .0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .245     0%    500  1532k     0  119k     0 1288k
09:30:01   5.0  .0 1.0  .0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .239     0%    500  1532k     0  119k     0 1288k
09:31:01   6.0  .0 1.0 1.0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .236     0%    600  1532k     0  119k 66505 1288k
09:32:01   5.0  .0 1.0  .0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .230     0%    500  1532k     0  119k     0 1288k
09:33:01   6.0  .0 1.0 1.0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .223     0%    600  1532k     0  119k 66505 1288k
09:34:01   5.0  .0  .0 1.0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .219     0%    500  1532k     0     0 66505 1294k
09:35:01   6.0  .0  .0 2.0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .221     0%    600  1532k     0     0  185k 1294k
09:36:01   7.0 1.0  .0 2.0 4.0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0  .0    .221     0%   3600  1532k  2863     0  185k 1294k
```

Figure 8-12 FXC145 Scheduler Queue Lengths by Time (SCHEDLOG)

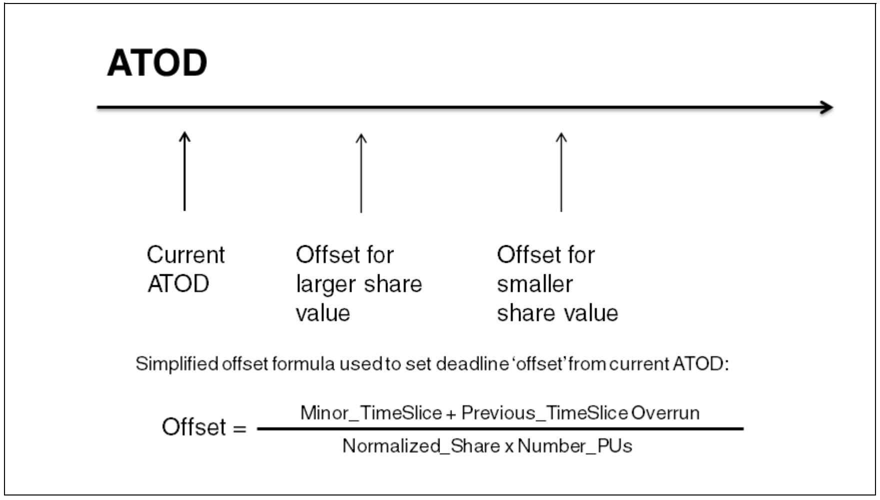
Next we look at the absolute and relative shares. The sum of absolute shares is 0% and the sum of the relative shares is 3600. Note the working set size for the work waiting to be dispatched. The Total Consid is the current working set size, 1532 K. We can see that 2863 is from Q0, 119 K for Q1, 456 k for Q2, and 898 K for Q3. The system evaluates this information to determine when put work in the Eligible Queue.

The size of the share is both a business decision and a performance decision. For example, if you assign one server a very high share, make sure that the server is critical to your business. This server can take resources as it needs them, but it could consume all allocated resource if it starts looping. Use this rule to allocate shares. Determine which servers need to run to meet your mission goals even when there is heavy contention for the processors.

TCP/IP, generally key in the network infrastructure, is an obvious candidate, as are required services such as Domain Name System servers. Give TCP/IP and other required servers absolute values that approach CPU consumption required at peak loads. There is no advantage to allocating more than that value. Analyze your peak loading performance data and periodically review system settings to maintain high performance.

8.6 ATOD deadline versus consumption

A deadline is a processor-computed priority that defines when a unit of work should be complete. A deadline is a time value on an artificial time of day (TOD) clock often referred to as an artificial time of day (ATOD). The deadline is computed based on several factors, but the most significant is the normalized share value, discussed in 8.5, "CP set share command" on page 147. This makes the share setting a significant system control. Virtual processors get ordered for dispatching based on their deadlines. Using Vm64721 and **`set srm limithard consumption`**, limit shares are controlled via a consumptions scheduler instead of a deadline scheduler. A deadline is the current default methodology, but VM is moving towards a consumption methodology. Figure 8-13 shows ATOD clock details.


ATOD
Current ATOD
Offset for larger share value
Offset for smaller share value
Simplified offset formula used to set deadline 'offset' from current ATOD:

$$\text{Offset} = \frac{\text{Minor\_TimeSlice} + \text{Previous\_TimeSlice Overrun}}{\text{Normalized\_Share} \times \text{Number\_PUs}}$$

Figure 8-13 ATOD calculation

8.7 Virtual Machine Resource Manager

Virtual Machine Resource Manager (VMRM), introduced in z/VM 4.3, runs in a service virtual machine (VMRMSVM) and dynamically manages the workload performance. VMRM uses a set of user-specified workload definitions and goals, compares these with the achieved performance, and makes adjustments accordingly. It is conceptually similar to the z/OS Workload Manager. The concept is to allow performance objectives to be set in a manner more closely aligned with business objectives than previously possible.

VMRM is only effective in a constrained environment because it works by prioritizing workloads to enhance their access to resources, and accordingly restricts other user access to those resources. If the z/VM system is not constrained, VMRM cannot enhance access to resources because they are readily available anyway.

A workload is a collection of virtual machines that are treated as an entity for performance management. Each workload has certain CPU utilization and defined DASD goals. It can also have a DASD goal, a CPU goal, or both. A goal represents the relative importance of a workload for the particular installation. You can change the configuration any time, but then need to stop and restart the VMRM service machine to activate your changes.

VMRM uses z/VM MONITOR data to obtain regular measurements of virtual machine consumption. VMRM then determines the level of activity on the system and whether workloads are meeting the defined goals. VMRM allows some latitude, 5%, in making this determination, which helps to avoid overreaction when a workload is near its goals.

The default and recommended setting for the MONITOR SAMPLE interval is one minute. Every minute, VMRM examines the system and workloads to find a workload not meeting its goals. VMRM then uses `cp set share` and `cp set iopriority` commands to adjust the workload, giving it more access to the resource for which the goals were not met. VMRM remembers which workload was adjusted recently and looks for another one to adjust, choosing a different workload to adjust at the next interval.

Note: We recommend a sample monitor interval of at least one minute but no greater than five minutes. A sample interval of less than one minute does not provide enough monitor data information for VMRM. Intervals greater than five minutes might cause VMRM to adjust workload performance settings too slowly.

VMRM cannot adjust users with fixed CPU or I/O priorities, so any such users on your system must be changed. This also allows specific users that otherwise would be treated as part of a workload to be effectively excluded from it. We recommend that you not define such users as part of a workload.

8.7.1 VMRM implications

VMRM can adjust to changing workload conditions because it adjusts performance parameters at frequent intervals. As the workload on the z/VM system changes, VMRM makes adjustments to the defined and running workloads to help meet defined goals. On a fully constrained system, VMRM cannot ensure workload goals because it cannot create resources. VMRM might conflict with manual efforts to tune z/VM because it adjusts z/VM tuning parameters. You might encounter problems, or unclear results, if you manually set `cp set share` and `cp set iopriority`. z/VM now supports I/O priority queuing and VMRM, a significant enhancement. `cp set iopriority` and the associated directory statement were added to z/VM to support VMRM.

VMRM uses `cp monitor` data so that it can be affected by other `cp monitor` data users, and vice versa. Remember this during setup. VMRM can make benchmarking difficult because it dynamically makes changes in response to varying conditions. Benchmarking typically requires repeatable conditions, and VMRM's continual adjustments make repeatable conditions unlikely.

8.7.2 VMRM Cooperative Memory Management

VMRM Cooperative Memory Management (VMRM-CMM) works a z/VM system and Linux guests to manage system memory constraints. Based on variables in the system and storage domain CP Monitor data, VMRM detects when there is a constraint and notifies specific Linux virtual guests. The guests can then take action to relieve the memory constraint.

VMRM provides Linux guests with workload management for CPU and DASD I/O, plus these functions:

- A NOTIFY statement with a MEMORY keyword in the VMRM configuration file that identifies the user of constrained memory.
- System and storage domains are monitored in order to calculate memory constraints and how much memory a guest should release.
- VMRM issues `cp smsg` to notify the guest how much memory to release to relieve the constraints.
- A message is logged in the VMRM log file with the user information and the text of the SMSG message.

Note: VMRM Cooperative Memory Management differs from Collaborative Memory Management Assist (CMMA), a function introduced in z9® hardware to support exchange memory management usage and status information with z/VM.

To learn more about VMRM, see Chapter 17 of the *z/VM Performance manual, Order Number*, SC24-6208.

8.8 Processor time accounting

Traditionally, Linux ran as the exclusive operating system on a discrete, distributed hardware server, and its built-in requirements reflected this role. When Linux runs on a mainframe, it becomes a virtual server environment that sits on other virtual layers. This is a fairly radical shift, one that requires fundamental changes.

8.8.1 Tick-based accounting

Older Linux kernels use a tick-based processor time accounting mechanism. On Linux, a tick, or a time interrupt, occurs every 1/100th of a second. Linux then learns exactly in which context the interrupt occurred, and accounts the entire time slice to this context. For example, in a a kernel interrupt, the entire time slice is accounted as system time.

In Figure 8-14, the top bar shows how Linux should account. The accounted figures belonging on the first bar of 2/100th seconds for kernel and 7/100th seconds for the user are correct. But in a real system the counts are made whenever the 1/100th second interrupt occurs. Depending on the time that the ticks begin, the reported numbers vary in the level of deviation from the true values. In the second bar, 1/100th seconds are counted on the kernel and 8/100th seconds are counted on the user context. The system clock is shifted in bar three where 4/100th seconds are counted on the kernel and 5/100th seconds are counted on the user context.

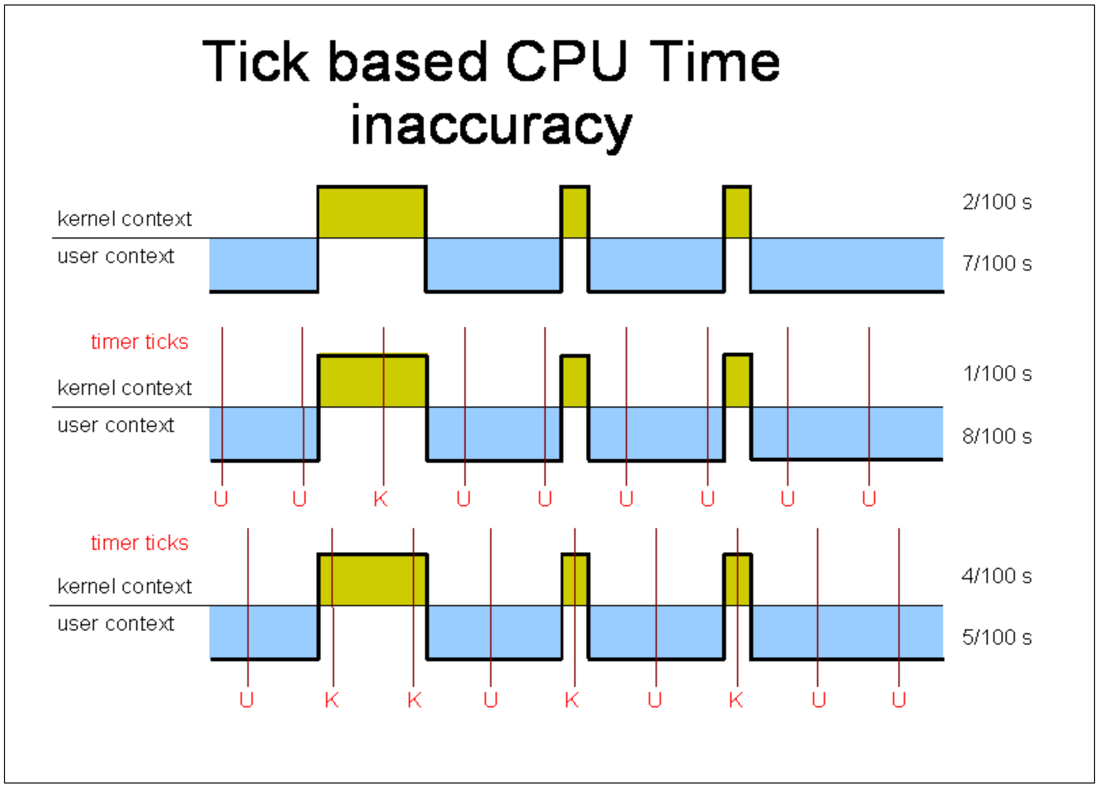


Figure 8-14 Tick-based processor time inaccuracy in a non-virtualized environment

Tick-based time accounting is not precise by nature, but it is effective when running on free-standing, non-virtualized servers. Over time, any inaccuracies will be balanced out.

However, the situation is different when running on z/VM or other virtualized environments, where Linux runs on virtual processors and where more virtual than real processors might be used. In this case, the time slicing concept skews Linux time sensing because Linux believes that it has control of the virtual processor at all times. In fact, the real processors can dedicate part of their time slice servicing another virtual processor, while the time slice might be accounted to a process that did not actually use it. This can lead to processor utilizations reported by Linux that can deviate from the load numbers reported by a virtual platform hypervisor. For example, when a user issues the **`top`** command, the resulting utilization numbers can vary considerably from the true values.

Figure 8-15 shows this virtualized environment difference. Figure 8-15 displays the time of involuntary wait states (that is, steal time), which is defined as the time that Linux wanted to run, but the hipervisor was not able to schedule a processor. This can happen in both z/VM and LPAR environments. The steal time is represented by the white space gaps in the shaded horizontal bars after the context is switched to the kernel or to the user.

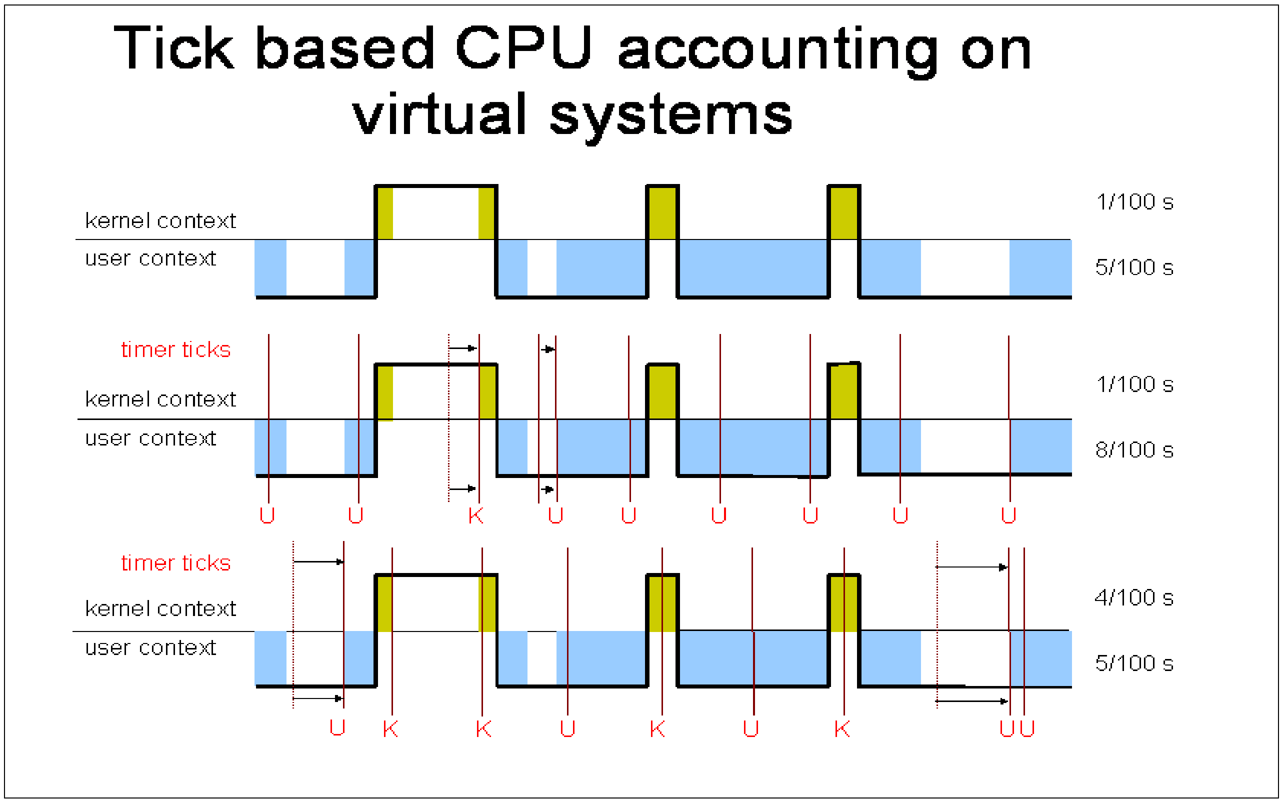


Figure 8-15 Tick-based processor accounting

As we mentioned before, the accounting to kernel or user differs significantly depending on the shift of the timer ticks. While the real values are one tick for the kernel context and five for the user context, the tick-based CPU accounting reports different kernel context numbers (for example, four in the third bar) or different user context numbers (for example, eight in the second bar), or both.

8.8.2 Virtual processor time accounting

The accounting mechanism must distinguish between real and virtual processor time, and recognize an involuntary wait state in order to improve processor time accounting accuracy. A new processor time accounting, virtual CPU time accounting, has been implemented for Linux on the System z platform with kernel 2.6.11. It is based on the System z virtual CPU timer instead of using wall-time as in the tick-based processor accounting. Now each processor has its own timer. The stepping rate is synchronized with the system TOD clock. Accounted time is incremented only when backed up by a physical processor. The Store CPU Timer (STPT) instruction is added to the Linux system call path. Linux can then calculate the user processor time by storing the CPU timer. CPU time accounting accounts processor time every time that the context changes, a more precise method than timer tick-based accounting.

Figure 8-16 shows details for CPU accounting.

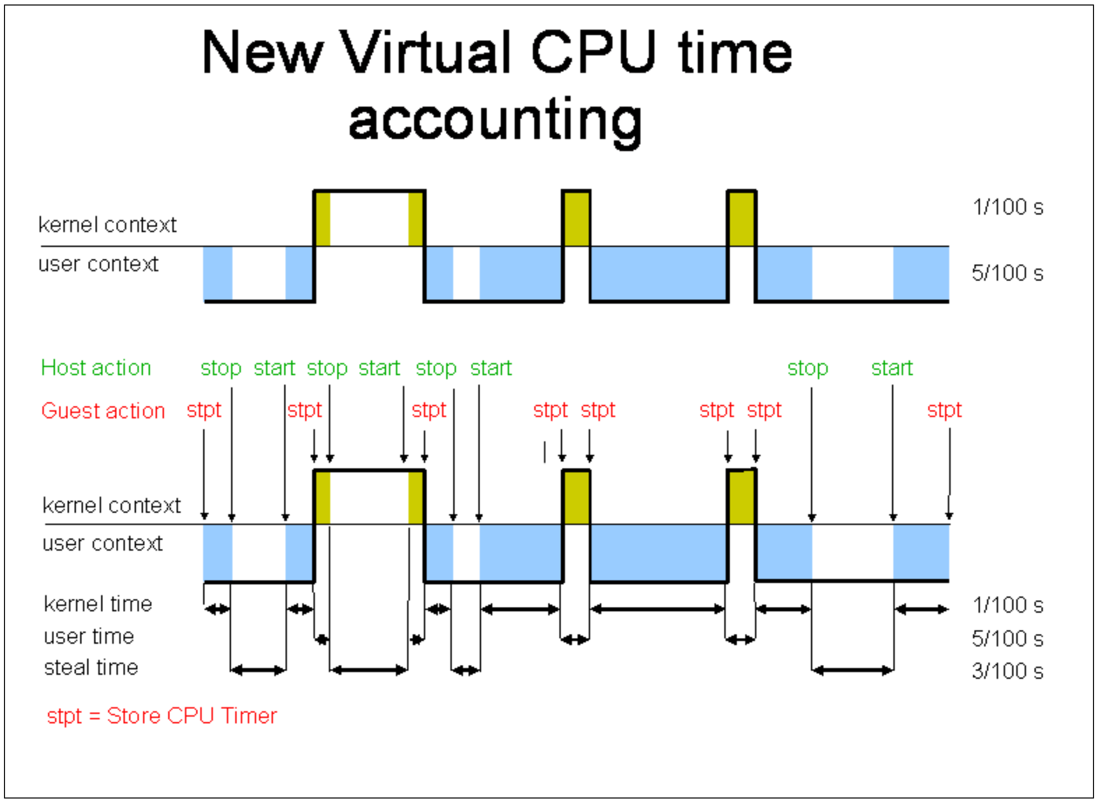


Figure 8-16 New virtual processor time accounting

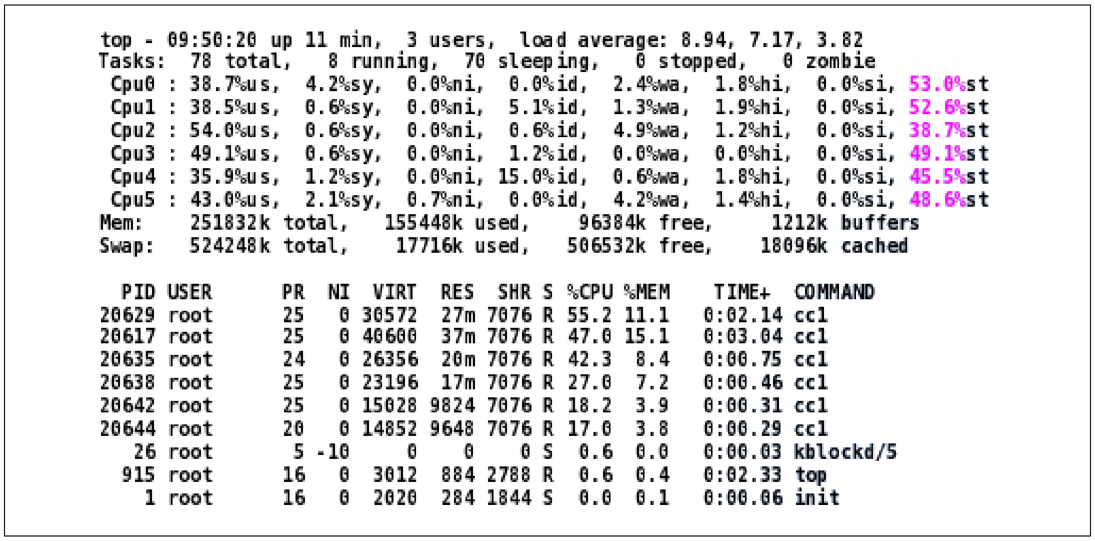
Recent Linux systems, beginning with SUSE SLES10 and Red Hat RHEL5, use the virtual processor accounting by default, but tick-based accounting is still available by configuring the kernel. With CPU time accounting enabled, Linux displays precise processor utilization times. These numbers are based on this:

- Tick-based processor time accounting reports processor time spent in a virtual processor context.
- Virtual processor time accounting reports processor times spent in real processor context.
- Internal precision of the processor time numbers, 1 microsecond on a System z machine.

Current versions of many tools display steal time if virtual CPU time accounting is active. Steal time is the percentage of time a virtual processor waits to get backed up by a real processor, while the hypervisor is not scheduling the virtual processor. Steal time is added to processor idle time if timer tick accounting is active. A prominent tool is the sysstat utilities package, which is often used to solve performance issues. The package has been updated to show steal time, as shown here:

- sysstat 6.0.2: mpstat and iostat display processor steal time.
- sysstat 7.0.0: mpstat, iostat, and sar -u display processor steal time.

Figure 8-17 shows output of the **top** command on Linux on System z with the new virtual processor time accounting implemented. The steal time is shown in the far right column.



```
top - 09:50:20 up 11 min,  3 users,  load average: 8.94, 7.17, 3.82
Tasks:  78 total,   8 running,  70 sleeping,   0 stopped,   0 zombie
 Cpu0 : 38.7%us,  4.2%sy,  0.0%ni,  0.0%id,  2.4%wa,  1.8%hi,  0.0%si, 53.0%st
 Cpu1 : 38.5%us,  0.6%sy,  0.0%ni,  5.1%id,  1.3%wa,  1.9%hi,  0.0%si, 52.6%st
 Cpu2 : 54.0%us,  0.6%sy,  0.0%ni,  0.6%id,  4.9%wa,  1.2%hi,  0.0%si, 38.7%st
 Cpu3 : 49.1%us,  0.6%sy,  0.0%ni,  1.2%id,  0.0%wa,  0.0%hi,  0.0%si, 49.1%st
 Cpu4 : 35.9%us,  1.2%sy,  0.0%ni, 15.0%id,  0.6%wa,  1.8%hi,  0.0%si, 45.5%st
 Cpu5 : 43.0%us,  2.1%sy,  0.7%ni,  0.0%id,  4.2%wa,  1.4%hi,  0.0%si, 48.6%st
Mem:    251832k total,   155448k used,    96384k free,     1212k buffers
Swap:   524248k total,    17716k used,   506532k free,    18096k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
20629 root      25   0 30572  27m 7076 R 55.2 11.1   0:02.14 cc1
20617 root      25   0 40600  37m 7076 R 47.0 15.1   0:03.04 cc1
20635 root      24   0 26356  20m 7076 R 42.3  8.4   0:00.75 cc1
20638 root      25   0 23196  17m 7076 R 27.0  7.2   0:00.46 cc1
20642 root      25   0 15028 9824 7076 R 18.2  3.9   0:00.31 cc1
20644 root      20   0 14852 9648 7076 R 17.0  3.8   0:00.29 cc1
   26 root       5 -10     0    0    0 S  0.6  0.0   0:00.03 kblockd/5
  915 root      16   0  3012  884 2788 R  0.6  0.4   0:02.33 top
    1 root      16   0  2020  284 1844 S  0.0  0.1   0:00.06 init
```

Figure 8-17 Sample of new TOP output

High numbers for steal time are not advantageous, especially over time. One result is that the application might be unresponsive, especially during online transactions. You can moderate such unwanted results by adding resources or modifying applications. These are reasons for accounting steal time:

- Processors are overcommitted by a too high ratio.
- Time is spent in the z/VM control program.
- Linux share per virtual processor is not large enough.
- Linux issues a high number of diagnose 44 or 9C instructions, which points to heavy context switching.
- Linux guest network I/O through a VSWITCH.

9

Tuning processor performance for z/VM Linux guests

In this chapter we discuss tuning processor performance for Linux guests, including recommendations, performance factors such as idle servers, the cpuplugd service, CPU topology support, and how to offload workloads to peripheral hardware.

9.1 Processor tuning recommendations

Processor time is a limited resource and, while some resource loss is unavoidable, you need to keep your system well tuned for optimal performance. We suggest that you routinely complete these steps:

- Eliminate unnecessary Linux services from the startup sequence. Default Linux guest installations typically start services that are not used but that do consume CPU cycles even if no work is performed.
- Remove unneeded tasks started by cron.
- Remove unnecessary process monitors that run at closely scheduled intervals and consume CPU cycles.
- Reduce processor usage for idle Linux guests.
  - 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.
- Prioritize workloads. Use share options to determine which workload should be completed first during processor constraints; other resources are often underutilized during constraints. Evaluate all option to reduce processor requirements and then take action to eliminate the bottleneck. There is little point in tuning other subsystems when the processor is already overcommitted.

9.2 How idle servers affect performance

There is no room for unnecessary processes in a shared resource environment. Redesign any servers that run cron jobs for historical reasons.

z/VM is a shared resource environment in which virtual machines are tailored for optimal performance. One of the most expensive uses of resources in a virtual environment is waking up an idle server simply to perform a trivial task. Pinging a server to see whether it is alive keeps the server active. Also, simple checks to confirm whether particular processes are active can cause a large, cumulative drain on CPU. Both these practices (unnecessary interrupts and queries) use resources that could be better utilized by other servers performing real work and detract from available resources. Always monitor your system for this type of expense and take steps to minimize it.

Routine pinging and queries are two common mistakes made in a virtual server infrastructure. You do not need to ping applications to ensure that they are alive or to monitor server performance. Usually idle servers do not use significant amounts of storage or processor resources. But when the applications are pinged or the servers are monitored, the server must have resident storage to provide correct responses. A better way to monitor your system is to use the existing and mostly free (in terms of resource requirements) VM monitor. Figure 9-1 displays effects of unnecessary daemons on the CPU and DASD.

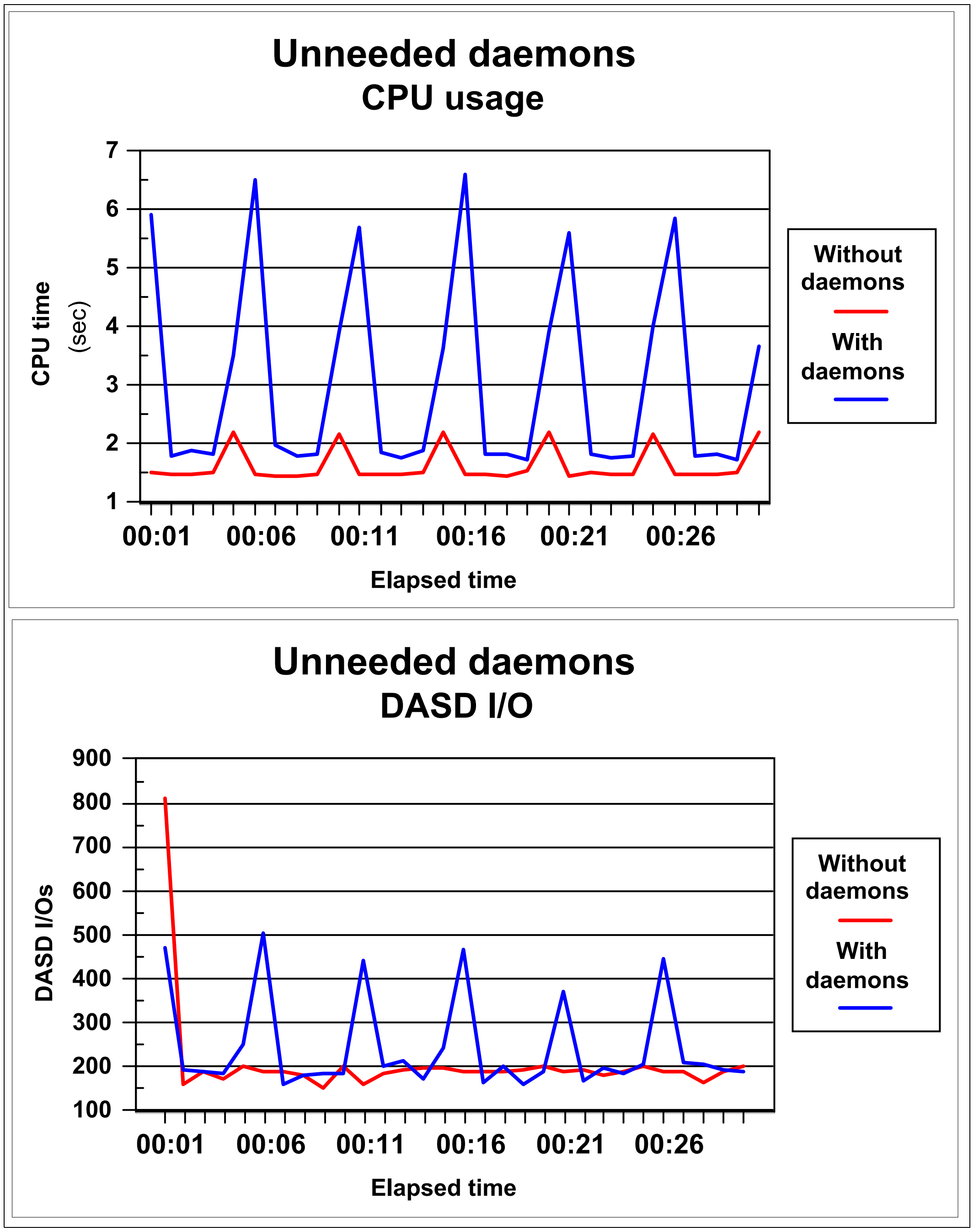


Figure 9-1 The effect of unneeded daemons on CPU and DASD usage

These are unnecessary services for a Linux guest:

- sendmail

  If Linux does not require the mail feature, consider stopping the sendmail server.
- anacron, atd, and cron

  These daemons initiate tasks at regular intervals. If there are no useful tasks automatically scheduled, stop these services.
- autofs, nfs, nfslock, and portmap

  The autofs daemon automatically mounts file systems. Nfs and nfslock provide Network File System (NFS) support, and portmap provides the Remote Procedure Call (RPC) services required by NFS.
- lpd and xfs

  lpd provides printing services, and xfs provides X-Windows fonts.
- inetd/xinetd

  inetd (or the replacement, xinetd) manages internet services such as FTP and Telnet.
- resmgr

  resmgrd is the resource manager daemon that allows applications to lock and access device files.

This list is not a complete one and is intended as an example. Inspect your system running services to detect unneeded resource consumers.

9.3 Performance effect of virtual processors

You can match System z resources to anticipated workloads by assigning virtual processors to a Linux virtual machine, but note that overall throughput can be affected by the number of processors assigned to Linux.

9.3.1 Assigning virtual processors to a Linux guest

CP manages all real processors defined to an z/VM LPAR. All virtual machines appear to have at least one processor that is referred to as the base virtual processor. You can add additional virtual processors to a virtual machine with the `cp define cpu` command. Virtual processors share real processor resources (Figure 9-2).

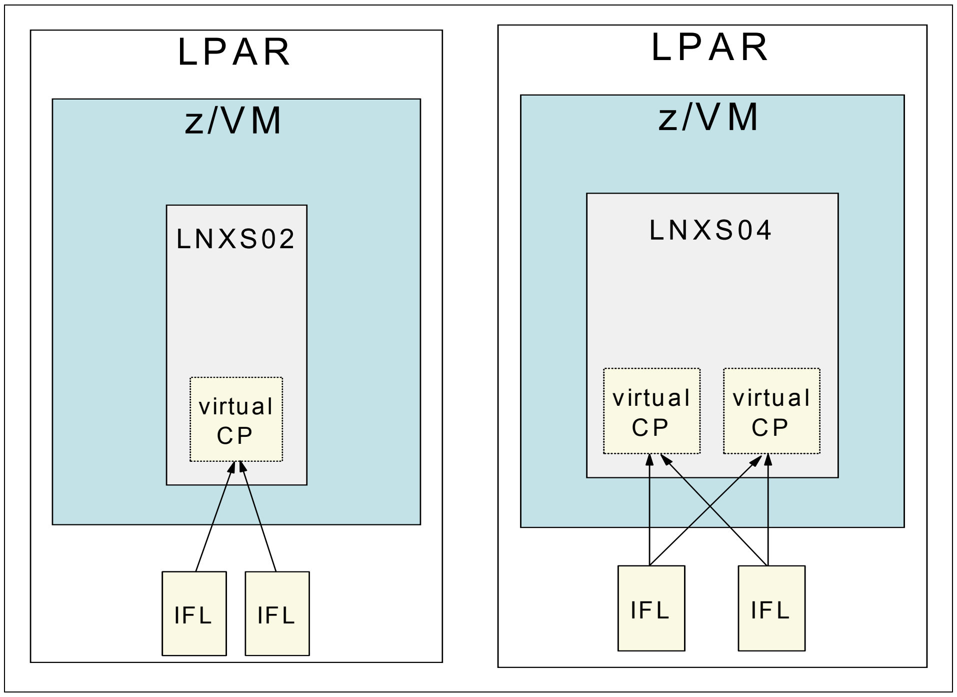


Figure 9-2 Virtual processors in a virtual machine

In Figure 9-2, Linux guest LNXS02 has access to two physical IFLs but, because it is a virtual uniprocessor, only one IFL can run at a time. Linux guest LNXS04 has two defined virtual processors, each using the physical IFLs in LPAR.

9.3.2 Measuring the effect of virtual processors

You can improve performance for processor-constrained workloads by adding virtual processors to a virtual machine. Use the WebSphere Performance Benchmark Sample workload to examine the effects of defining virtual processors to a Linux guest. We used the WebSphere Performance Benchmark Sample in a three-tier configuration to determine the relative cost of processor resources for each workload component, as follows:

- WebSphere Application Server
- HTTP server
- DB2 server

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

Figure 9-3 displays the effect that varying the number of virtual processors has on CPU utilization. In Figure 9-3, 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 to four.

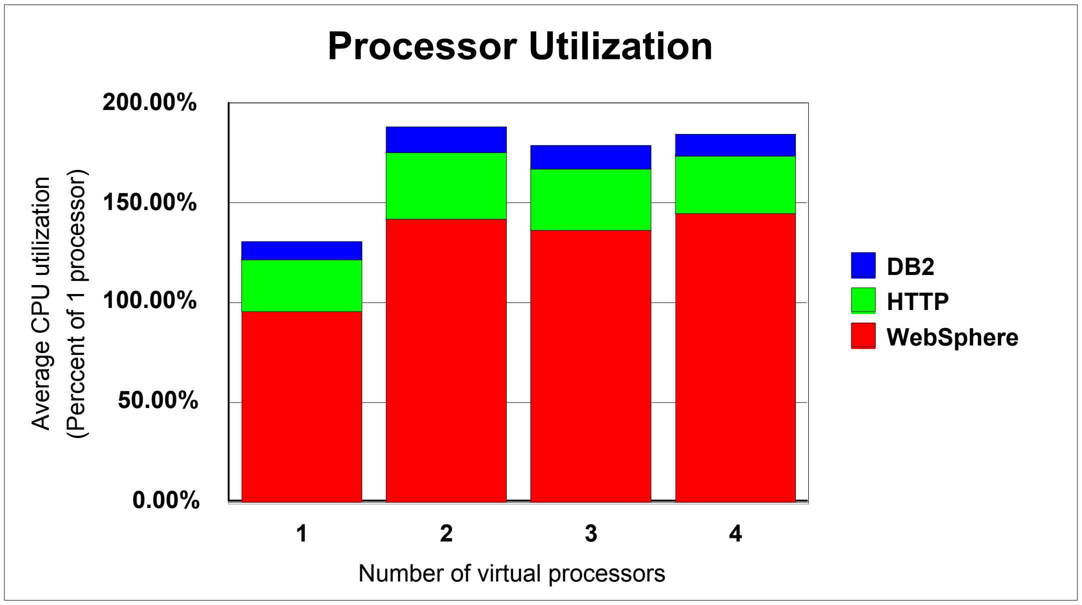


Figure 9-3 Measuring the effect of virtual processors on CPU utilization

You can see that 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.

We arrived at the cost of adding virtual processors by using the average response time and average CPU time utilization. We calculated the average cost of a WebSphere Performance Benchmark Sample transaction, measured in milliseconds per transaction, by using the reported transaction rate and measured average CPU time expended in each Linux guest (Figure 9-4).

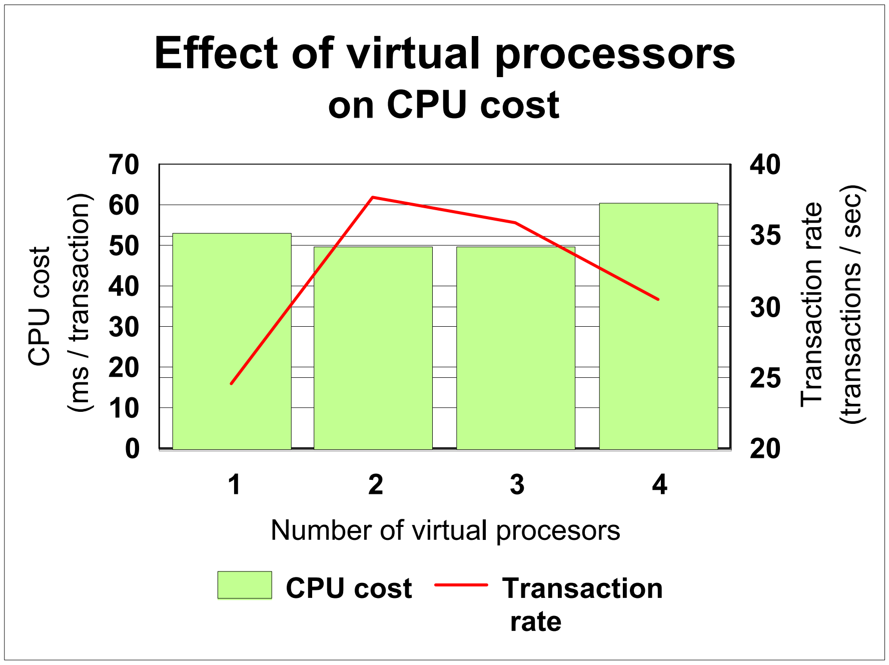


Figure 9-4 Measuring the cost of adding virtual processors

In Figure 9-4, we see that the average transaction cost decreases slightly when the number of virtual processors matches the number of real processors. More importantly, the average transaction rate increases significantly. Note that as the number of virtual processors increases 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.

We recommend the following guidelines whenever Linux runs a processor-constrained workload:

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

9.3.3 Managing processors dynamically in Linux (cpuplugd service)

The cpuplugd service controls the number of CPUs for Linux by increasing or reducing the amount of available CPUs while Linux is running. You can adjust the minimum and maximum range of CPU and the value when CPUs are switched on or off. The cpuplugd service allocates the CPUs needed to perform the workload.

We discussed adapting the number of processors to the workload in paragraph 9.3.2, “Measuring the effect of virtual processors” on page 161. An advantage of the cpuplugd is that processors are not idling, and therefore have a loaded run queue. This reduces the amount of processor signals needed to wake up the system.

We use a modified dbench benchmark, running a fileserver workload, to show cpuplugd service advantages. Many memory operations of workloads are of mixed file operations, for example, creating, writing, reading, appending, and deleting. In this test, we used the dbench benchmark that was modified by creating two processes per client and that communicated with POSIX messages queues. The first process reads I/O commands, and the second executes the commands. This is not advisable if you have many idle CPUs, because while a message is sent from the first to the second process, the CPU waiting on the second process must be wakened. This costs a lot of time. You can save time if CPUs are busy because there is no need to waken any CPUs. Figure 9-5 shows dbench throughput. The x axis number shows started dbench processes, whereas the y axis shows the dbench throughput in MB per second. In our example, the x axis shows that the started processes doubled due to our modifications. Our base measurement is the red line, which shows measurement without cpuplugd service, indicating that all CPUs were available during the experiment.

The green line shows a second measurement with the cpuplugd service, defined with two online CPUs as the minimum and 20 CPUs as the maximum. The load was checked every second. When we compare the green and red lines, we see that throughput is better running up to 26 dbench processes. The green line throughput is up to 40% better (for example, take 12 parallel dbench processes).

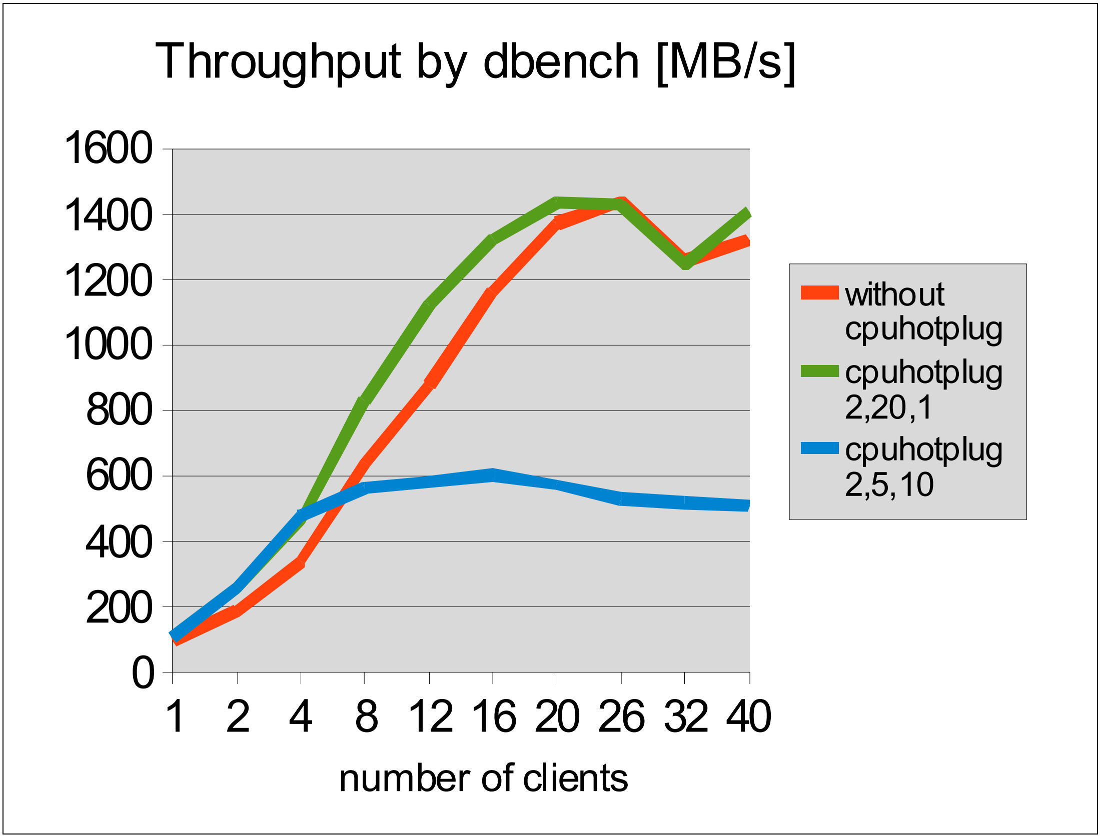


Figure 9-5 Throughput measured with and without cpuplugd service

Figure 9-6 shows the processor cost savings per transferred MB. The x axis shows the started dbench processes, and the y axis shows the cost savings.

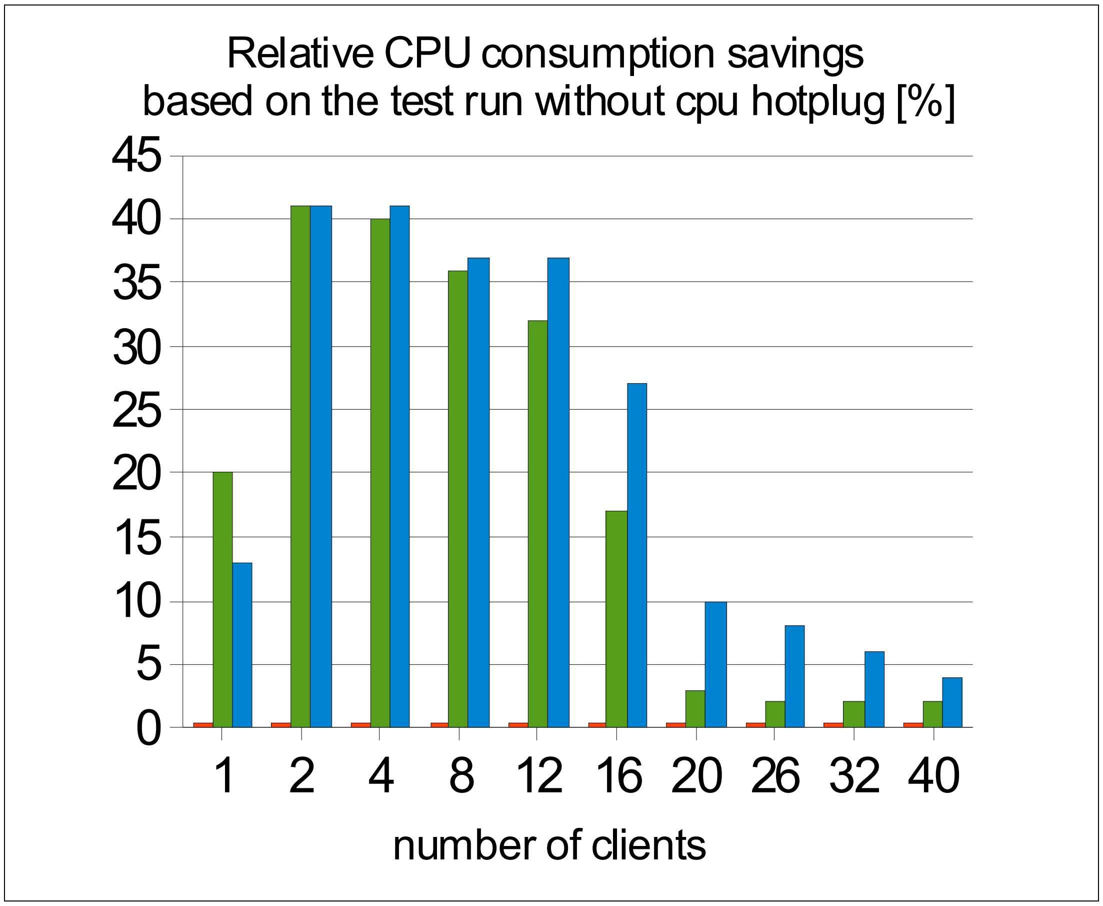


Figure 9-6 Cost savings relative to the non cpuplugd service measurement

There are processor savings of up to 40%, especially when fewer Linux processes are running in parallel. Costs are the same level when all 20 available CPUs are online. Using the cpuplugd service is ideal in an environment where CPUs would be idling without cpuplugd.

cpuplugd service is available on all current distributions. Activate it by setting parameters in the Linux kernel configuration file (Example 9-1).

Example 9-1 Checking the Linux kernel configuration for cpuplugd service prerequisites

```
lnxwas:/usr/src/linux # uname -a
Linux lnxwas 2.6.32.12-0.7-default #1 SMP 2010-05-20 11:14:20 +0200 s390x s390x
s390x GNU/Linux

lnxwas:/usr/src/linux # zcat /proc/config.gz > .config

lnxwas:/usr/src/linux # cat .config | grep CONFIG_64BIT
CONFIG_64BIT=y

lnxwas:/usr/src/linux # cat .config | grep CONFIG_SMP
CONFIG_SMP=y
```

```
lnxwas:/usr/src/linux # cat .config | grep CONFIG_HOTPLUG_CPU
CONFIG_HOTPLUG_CPU=y
```

You need to define rules to adjust the cpuplugd service, that is, to increase or decrease the number of CPUs (Example 9-2). These rules are specified in the `/etc/sysconfig/cpuplugd` file and are based on constants and variables, for example, *loadavg* (current CPU load of the last minute), `onumcpus` (the actual number of CPUs online), `idle` (current idle percentage), and `runable_proc` (the current amount of runable processes). The rule HOTPLUG defines when additional processors are put online. Use the rule HOTUNPLUG to decrease the number of processors.

Example 9-2 cpuplugd service configuration is in /etc/sysconfig/cpuplugd (some lines deleted)

```
#
# Exemplary configuration file for the cpuhotplug daemon for
# Linux on System z
#
...
# The minimum number of cpus.
# This means in this example, that every time at least two cpus
# will be available
#
CPU_MIN="2"
...
# The maximum number of cpus to be enabled. If 0 is specified here,
# the maximum number of cpus equals to the number of cpus detected.
#
CPU_MAX="0"
...
# The update interval described how often the current system state is
# changed against the configured set of hotplug and hotunplug rules. The
# update intervall is defined in seconds.
#
UPDATE="10"
...
# The minimum size of the static page pool
#
CMM_MIN="0"
...
# Ruledefinitions
#
# Four kinds of rules are distinguished
#   (1) hotplug rules, used to enable cpus
#   (2) hotunplug rules, to disable cpus
...
# Within the hotplug/hotunplug rule definitions the following variables
# can be used:
#   - loadavg:          the current loadaverage
#   - onumcpus:         the actual  number of cpus which are online
...
#   - idle:             the current idle percentage
#
HOTPLUG="(loadavg > onumcpus + 0.75) & (idle < 10.0)"
#
```

```
HOTUNPLUG="(loadavg < onumcpus - 0.25) | (idle > 50)"
...
lnxwas:~ #
```

Select the update interval carefully. A value that is too low will defer processor configuration changes. A value that is too high is not desirable because changes in processor numbers produces overhead.

Setting cpu_min to 2 might be too high in certain cases. The cpumax value must be high enough to cover workload peaks, or must stay at default 0, which translates to all processors detected by the system.

9.3.4 CPU topology support

Current mainframes such as System zEnterprise or z196 provide different caches levels. Certain caches service only one processor, whereas others are shared by several or all processors. Our machines have closer caches, which are located on the same book. Others are remote located on other books.

The Scheduler usually runs a process on the same processor as the previous one, but, in certain situations, the Scheduler can migrate a process to another processor. This can be a result of workload balancing or when a processor is dynamically taken away from Linux. During migration, it is an advantage for the Linux Scheduler to have additional information about cache topology that allows it to reschedule to a processor using caches close to its origin.

z10 and newer machines support an interface than can obtain processor topology for a LPAR. Note that the CPU topology support is switched off by default. The Scheduler uses this information to determine which process will run on which processors. The information is kept in `/sys/devices/system/cpu/`*`cpu<N>`*`/topology/core_siblings`, where *cpu<N>* is the number of the processor in question (Example 9-3). The mask is shown as a hexadecimal number, here X’03’ or binary B’11’, indicating that CPU0 and CPU1 are neighbors.

Example 9-3 Display attribute core_siblings for CPU0

```
lnxwas:~ # cat /sys/devices/system/cpu/cpu0/topology/core_siblings
00000000,00000003
lnxwas:~ #
```

Topology support can even handle dynamic changes like machine reconfigurations. Offline processors are included in the mask if the kernel supports activation and deactivation of processors. Configuration adoptions generate a change event and can be detected. However, it might take time for dynamic changes to be reflected in the mask.

9.4 Offloading workload to peripheral hardware

Current distributions support various hardware features that can offload tasks to the peripheral hardware. Offloading reduces processor time needed on the mainframe processors that might then be used by other tasks and processes in Linux, by other LPARs or z/VM guests.

A widely available option for offloading is OSA-Express cards of various generations that offer offloading functions such as these:

- TCP segmentation offload or Large Send: Offloads TCP segmentation operation from the Linux network stack to the OSA-Express2 or OSA-Express3 features
- hw_checksumming: Redundancy check to protect data integrity

Another option is the cryptographic support provided by the Central Processor Assist for Cryptographic Function (CPACF) and Cryto Express2 (CEX2) features. CPACF supports several block ciphers and secure hash functions and executes requests synchronously to the processor. CPACF is a built-in feature in the mainframe. You need to correctly configure software to exploit the hardware. CEX2A is optional and can asynchronously execute cryptographic requests. CEX2A accelerates public key operations used for the SSL handshake and needs Linux driver support. You realize better throughput by combining CPACF and CEX2A. Learn more about cryptographic benchmarks at this link:

http://public.dhe.ibm.com/software/dw/linux390/perf/crypto.pdf

10

Tuning disk performance

This chapter discusses how you can best tune Direct Access Storage Device (DASD) and Small Computer System Interface (SCSI), including disk and access options, z/VM and System Z disk setup, influencing factors for I/O, and how to maximize disk operations.

10.1 Disk options for Linux on System z

You can choose disk devices from many options, such as DASD, SCSI, and z/VM controlled devices minidisk. All of these devices communicate through different disciplines and access methods.

10.1.1 DASD with ECKD or DIAGNOSE

Linux on System z supports DASD devices such as 3390 Mod 3, Mod 9, Mod 27, Mod 54, and Mod A. The largest size supported is raw 223.2 GB and approximately 180 GB formatted. Today these devices are emulated by storage subsystems and connect to DASD in various ways. FICON adapters offer quick access; Linux as z/VM offers guest DIAGNOSE access.

10.1.2 SCSI disk

The Fibre Channel Protocol (FCP) allows mainframes to attach to SCSI devices and also enables z/VM or Linux on System z to access these devices. This connectivity provides enterprises with a wide range of choices for storage solutions, including using existing storage devices. FCP channel support allows Linux instances on System z to attach and communicate with SCSI devices.

10.1.3 z/VM minidisks

z/VM minindisks are the standard method for allocating disk resources to users and guest systems like Linux. A large system volume, such as an emulated 3390-3, contains 3339 cylinders when formatted before being subdivided, of which 3338 are available for allocation to users. The first cylinder, numbered 0, is reserved for the Volume Table of Contents (VTOC). First, a new volume is prepared by formatting, and then is allocated to the user extent, giving it a type of PERM, denoting permanent user space. These prepared 3338 cylinders are allocated to users and typically split into portions of varying length, often by using the DIRMAINT tool. An entry is created in the user VM directory entry to define the minidisk (MDISK). If a directory management tool is not available, this entry can be prepared using the VM Xedit editor. The line defining a minidisk in the directory is shown here:

```
MDISK 0191 3390 61 20 LX6U1R MR
```

Reading the fields from left to right, this line entry is defining a MDISK, the virtual address, the emulated disk device type, the starting cylinder address, the size in cylinders, the volume name, and the mode or level of disk access privilege. MR denotes a conditional multiple-write mode.

10.2 Factors influencing disk I/O

Various factors can influence disk I/O and the times spent in different I/O stages from Linux to disk and vice versa. These factors include DASD I/O and SCSI I/O.

10.2.1 DASD I/O

You need to consider several components of response time when evaluating DASD I/O performance. Figure 10-1 shows some of these components, which are described in this section.

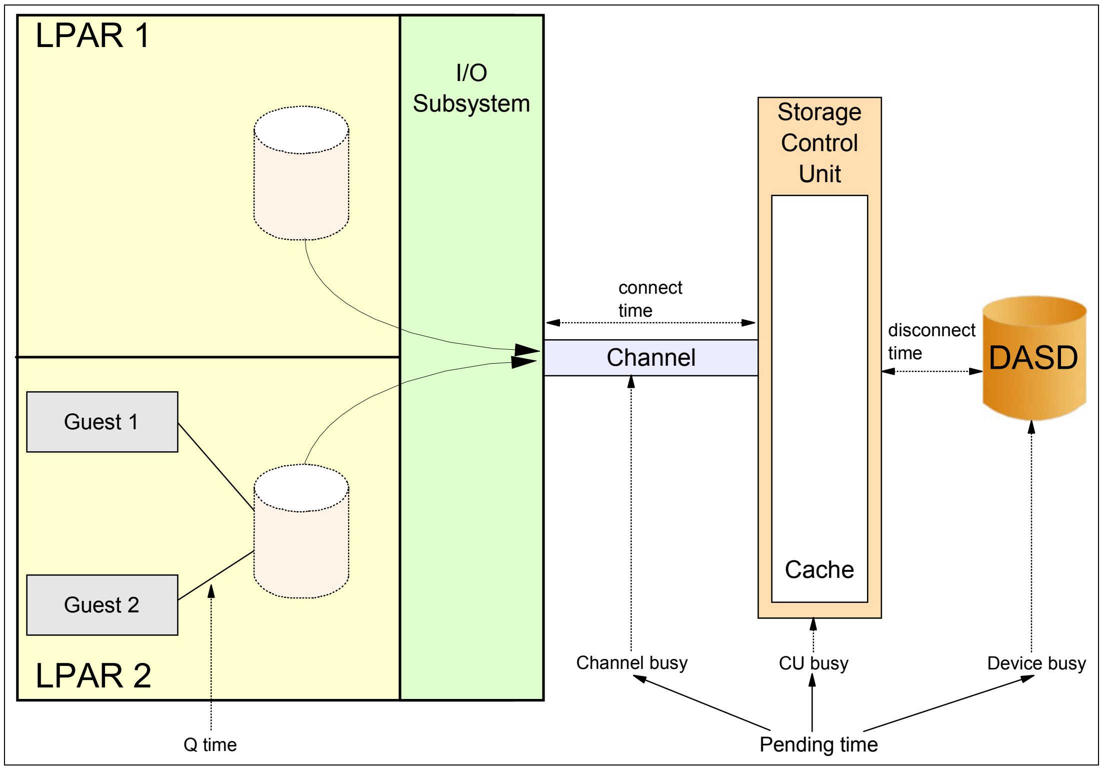


Figure 10-1 Components of overall DASD response time

These are DASD I/O response time components:

- Queue time

  Results when multiple users simultaneously access 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 extremely variable, and under heavy load, most serious. High queue times can indicate operating system contention, or a high device service time; as I/O requests take longer to complete, service requests from other users can cause queuing for heavily used devices.

- Connect time

  The actual time to transfer the data on the channel, normally less than 2 ms for a 4 K block of data.

- Pending time

  The time required to start I/O, normally less than 1 ms. High pending time reflects contention on the I/O path, possible due to contention on the channel, on the control unit, or on the DASD device.

- Disconnect time

  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 - 3 ms on cache controllers.

- Service time

  The sum of pending, connect, and disconnect times.
- Response time

  The sum of queue and service times.

10.2.2 SCSI I/O

You also need to consider several components of response time when evaluating SCSI I/O performance:

- Queue time

  If the device is busy servicing an I/O request, additional I/O requests wait in the queue. The length of time that 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 fabric latency. As I/O requests take longer to complete, requests for service from other users can cause queuing for heavily used devices.
- Fabric latency

  The time from entering the fabric until coming back, including all the time needed on the storage subsystem and in switches.
- Channel latency

  Additional time spent in the channel to fulfill the request.

10.3 Observing disk performance

Usually you monitor performance and identify performance issues using resource management tools. There are many ways to collect the data, for example, online tools, ECKD, and SCSI statistics.

10.3.1 DASD statistics

Linux collects performance stats on DASD activity as seen by Linux. The DASD driver carries out the data collection. Because the problem is generally with single, heavily accessed disks rather than with all disks, begin by viewing statistics for single DASD devices (Example 10-1).

Example 10-1 DASD statistics - How to use

```
Turn on with
echo set on > /proc/dasd/statistics
Turn off with
echo off > /proc/dasd/statistics
Reset:
echo reset > /proc/dasd/statistics
Can be read for the whole system by cat /proc/dasd/statistics
Can be read for individual DASDs by tunedasd -P /dev/dasda
```

For details, instructions and examples, see this website:

http://public.dhe.ibm.com/software/dw/linux390/perf/Linux_standard_monitoring_tools.pdf

10.3.2 SCSI/FCP statistics (SLES11 SP1, kernel 2.6.32)

With kernel 2.6.32, changes have been made to the way that these statistics are collected.

For comprehensive documentation of the available SCSI statistics values and how to interpret these values see Chapter 9 at the following link:

http://public.dhe.ibm.com/software/dw/linux390/docu/lk32ts03.pdf

You can access any or all of these values out of the `sys` file system manually with the **`cat`** command or by using your own scripts.

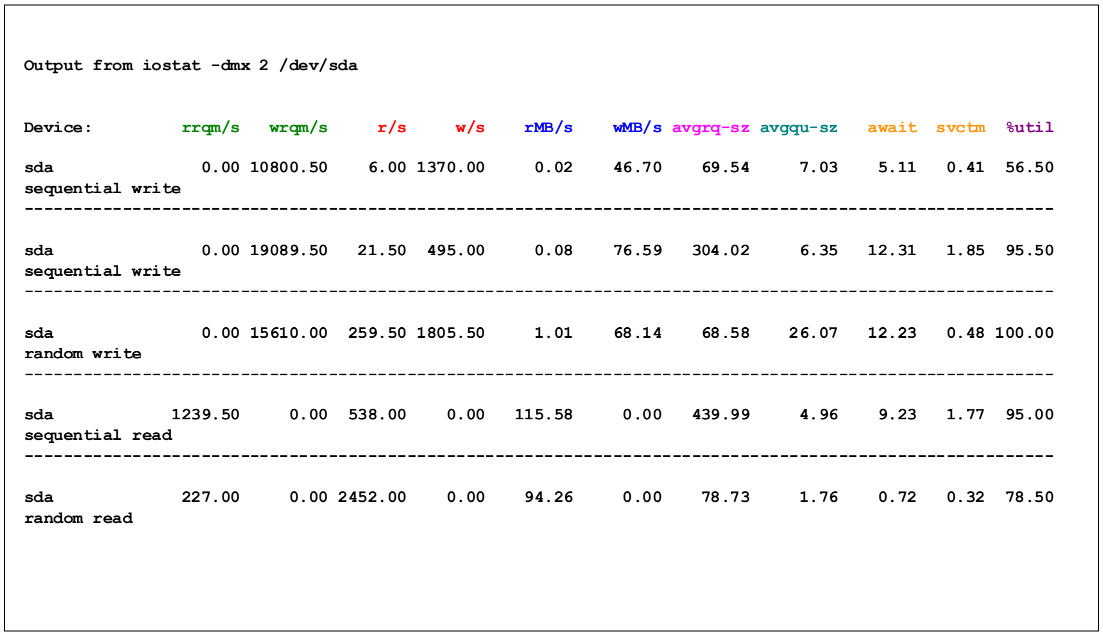
There is also the **`ziomon`** monitor tool, which can monitor zfcp performance data over a period of time and also provides input to capacity planing described in Chapter 12 of the documentation that can be found here:

http://public.dhe.ibm.com/software/dw/linux390/docu/lk32ts03.pdf

10.3.3 Monitoring disk statistics using iostat

iostat is part of the sysstat rpm. See 3.6.1, “System status (sysstat) tool” on page 32 for a comprehensive description of the sysstat package. The different queuing situations in ECKD and SCSI might lead you to the wrong conclusions if you view only specific statistics like DASD or z/VM monitor data on ECKD disk I/O. This is especially true for response times, because they represent round-trip times for single requests. All queuing in or in front of the DASD driver is not shown. Because there is not an equivalent to the SCSI subchannel serialization, zFCP statistics usually show longer round trip times per request than the DASD statistics. However, for SCSI there can be a queue with up to 32 requests in the storage server per LUN. Therefore, the only way to compare performance data of ECKD and SCSI is at the level of the Linux common I/O where iostat takes the numbers.

Data is directly available provided that you entered the correct command parameters. In Figure 10-2, we used the command **`iostat -dmx 2 /dev/sda`** to display requests merged per second. Figure 10-2 also shows I/Os per second and throughput per second. The average request size informs about record sizes, stripe sizes, or read ahead values. The queue size indicates how heavily the device is filled up. Await counts the average time for a request. Service time counts the portion to the storage server and back. Utilization indicates how busy a device is, and whether the data can be split and moved to another device.



```
Output from iostat -dmx 2 /dev/sda


Device:          rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz avgqu-sz   await  svctm  %util

sda                0.00 10800.50    6.00 1370.00     0.02    46.70    69.54     7.03    5.11   0.41  56.50
sequential write
----------------------------------------------------------------------------------------------------------

sda                0.00 19089.50   21.50  495.00     0.08    76.59   304.02     6.35   12.31   1.85  95.50
sequential write
----------------------------------------------------------------------------------------------------------

sda                0.00 15610.00  259.50 1805.50     1.01    68.14    68.58    26.07   12.23   0.48 100.00
random write
----------------------------------------------------------------------------------------------------------

sda             1239.50     0.00  538.00    0.00   115.58     0.00   439.99     4.96    9.23   1.77  95.00
sequential read
----------------------------------------------------------------------------------------------------------

sda              227.00     0.00 2452.00    0.00    94.26     0.00    78.73     1.76    0.72   0.32  78.50
random read
```

Figure 10-2 Example output for SCSI disk sda

Here are descriptions of the fields shown in Figure 10-2:

- Merged requests per second: rrqm/s (read) and wrqm/s (write).
- I/Os per second: r/s (read) and w/s (write).
- Throughput: rMB/s (read) and wMB/s (write).
- Average request size: avgrq-sz reflects the record sizes, stripe sizes and read ahead.
- Average queue length: queue size shows how much data traffic is going to the device.
- Service times: await shows the time for a request, and svctm the portion of it counted for the way to the storage server and back.
- Utilization: Reflects device utilization, and hints if content should be split onto different devices.

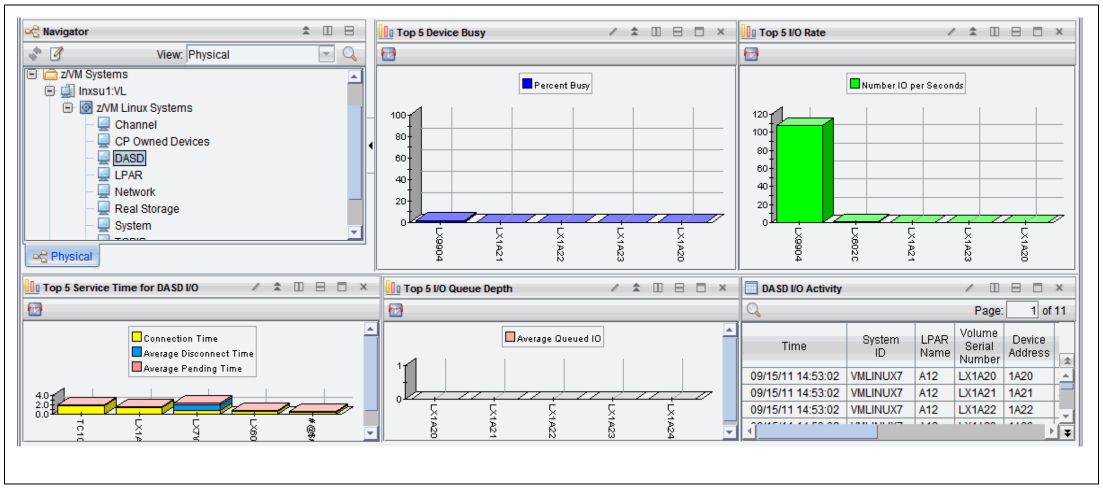
Note: When comparing latencies between ECKD DASD and SCSI LUN, keep in mind that DASD requests may get queued in the driver, whereas SCSI LUN requests may get queued both in the driver and in queues at the storage server. You cannot compare latency numbers directly because measurement results reflect the different characteristics.

10.3.4 Monitoring disk statistics with Omegamon and z/VM Performance Toolkit

This section discusses monitoring disk statistics with Omegamon and z/VM Performance Toolkit.

Omegamon

Figure 10-3 shows statistics available for each DASD volume. It is less critical for you to tune at this level because recent disk hardware has its own sophisticated management interface. But you still need to analyze and understand existing access patterns (for example, relative numbers of reads and writes at the device level) when you determine whether to implement features such as minidisk cache, which is active much further up the I/O chain and closer to the application. Device-level contention is typically lower on a well-architected system, but if problems arise, or you need the highest level of optimization, these statistics are useful. Figure 10-3 shows the OMEGAMON workspace listing the top five busiest disk volumes. Note that many more low-level details are available for every volume.



Time	System ID	LPAR Name	Volume Serial Number	Device Address
09/15/11 14:53:02	VMLINUX7	A12	LX1A20	1A20
09/15/11 14:53:02	VMLINUX7	A12	LX1A21	1A21
09/15/11 14:53:02	VMLINUX7	A12	LX1A22	1A22

Figure 10-3 OMEGAMON top five DASD volumes

Figure 10-4 shows the response time and overall I/O rate for a DASD volume. Average response time is just over one millisecond per I/O, which is relatively good for this system. The average I/O rate is 130.9 I/O per second. The output is from the IBM PAF tool.

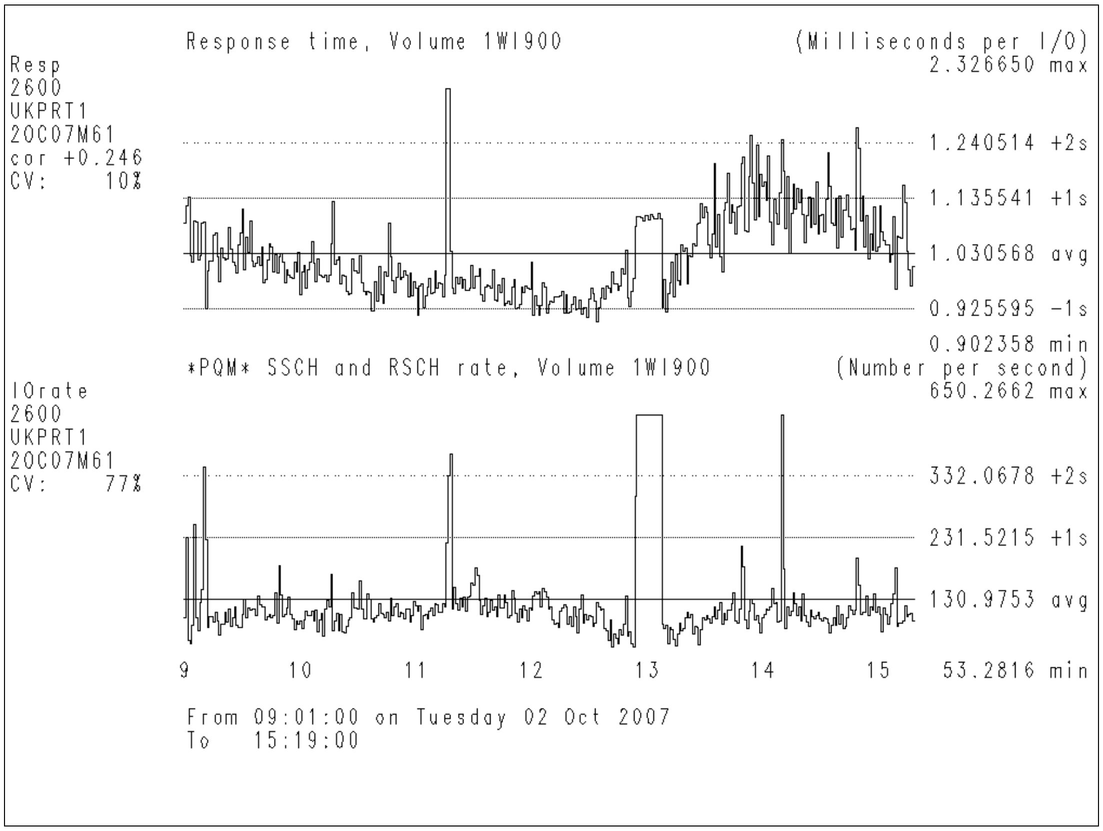


Figure 10-4 Response time and I/O rate chart for a single volume

Figure 10-5 displays a breakdown of the overall I/O response time for the same volume, showing pend time, disconnect, and connect time. Note that the pend and disconnect times are low in absolute terms, and relative to the connect time, the majority of the response time is spent transferring data to the disk, with no queue time.

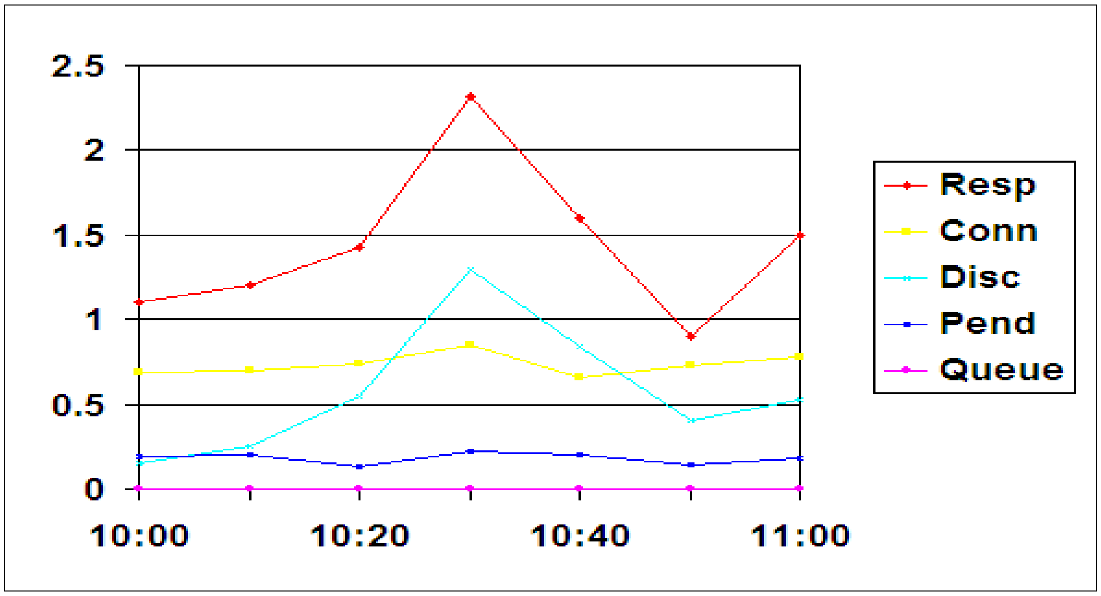
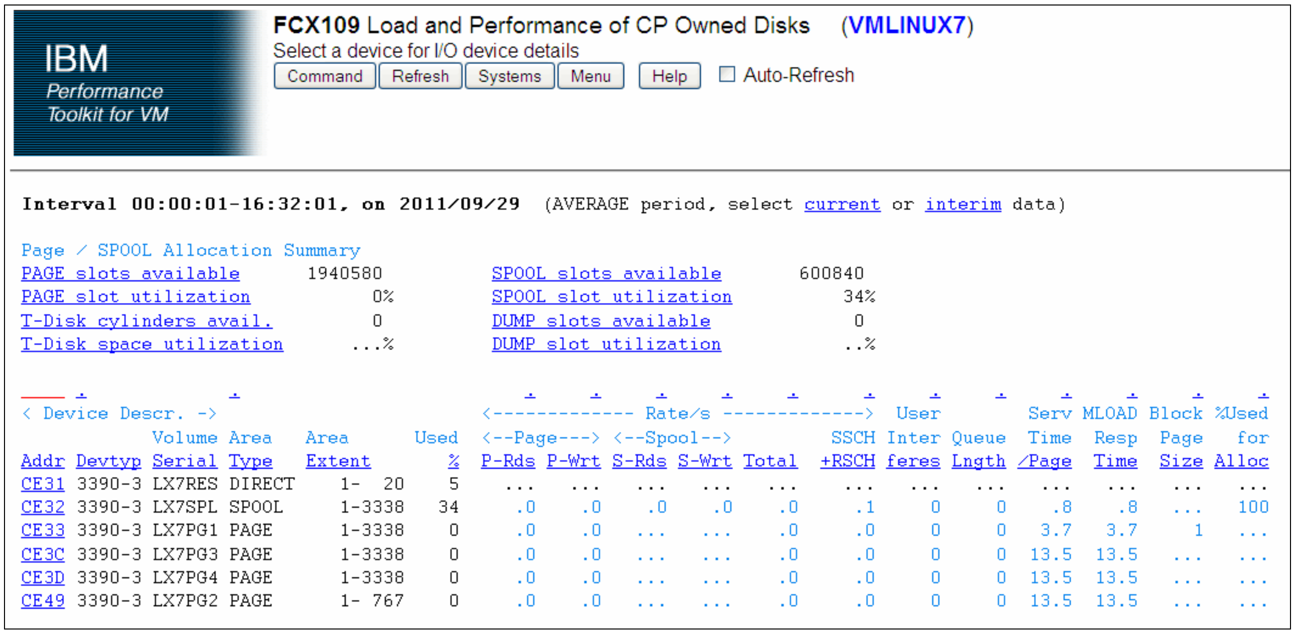


Figure 10-5 Overall I/O response time breakdown chart

z/VM Performance ToolKit

Examine the paging subsystem in case you need to add additional DASD to the paging configuration. Figure 10-6 outlines DASD paging devices located at address CE33 - CE49. The system is not paging out to DASD.



```
FCX109 Load and Performance of CP Owned Disks     (VMLINUX7)
Select a device for I/O device details

Interval 00:00:01-16:32:01, on 2011/09/29  (AVERAGE period, select current or interim data)

Page / SPOOL Allocation Summary
PAGE slots available       1940580         SPOOL slots available        600840
PAGE slot utilization           0%         SPOOL slot utilization          34%
T-Disk cylinders avail.         0          DUMP slots available            0
T-Disk space utilization     ...%          DUMP slot utilization          ..%

< Device Descr. ->                          <------------- Rate/s ------------->  User       Serv MLOAD Block %Used
            Volume Area    Area        Used <--Page---> <--Spool-->       SSCH Inter Queue  Time  Resp  Page   for
Addr Devtyp Serial Type    Extent         % P-Rds P-Wrt S-Rds S-Wrt Total +RSCH feres Lngth /Page  Time  Size Alloc
CE31 3390-3 LX7RES DIRECT    1-  20       5   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
CE32 3390-3 LX7SPL SPOOL     1-3338      34    .0    .0    .0    .0    .0    .1     0     0    .8    .8   ...   100
CE33 3390-3 LX7PG1 PAGE      1-3338       0    .0    .0   ...   ...    .0    .0     0     0   3.7   3.7     1   ...
CE3C 3390-3 LX7PG3 PAGE      1-3338       0    .0    .0   ...   ...    .0    .0     0     0  13.5  13.5   ...   ...
CE3D 3390-3 LX7PG4 PAGE      1-3338       0    .0    .0   ...   ...    .0    .0     0     0  13.5  13.5   ...   ...
CE49 3390-3 LX7PG2 PAGE      1- 767       0    .0    .0   ...   ...    .0    .0     0     0  13.5  13.5   ...   ...
```

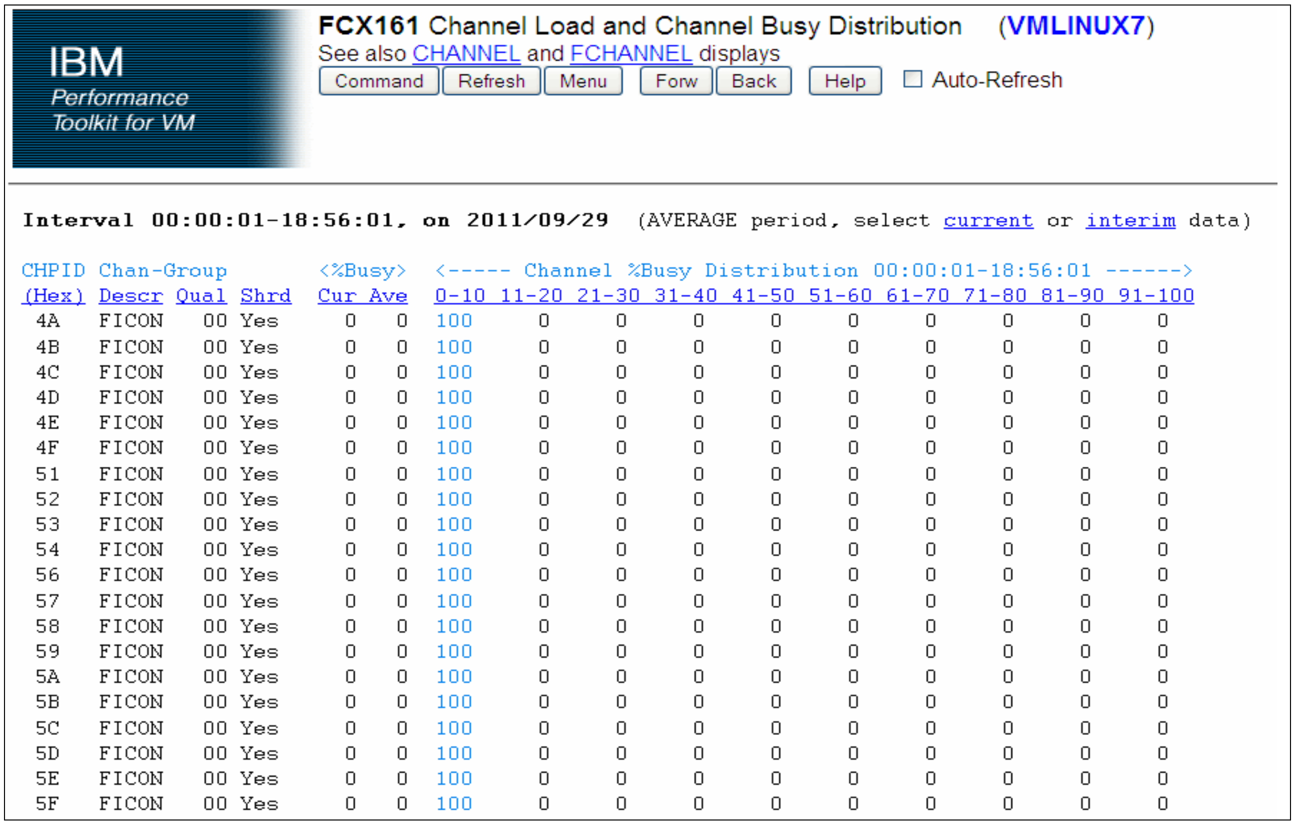
Figure 10-6 FCX109 CP owned disks

If you want to add paging devices to the configuration, you need sufficient channel capacity to support the additional devices. In this DASD configuration, the paging devices share four channel paths or CHPIDs. The devices are in the range of CE31 - CE49 and are on CHPIDs 52, 56, 5A, and 5E. Example 10-2 shows the configuration.

Example 10-2 FCX131 I/O device configuration

```
Status   08:47:23

<----  Ranges  ---->  Device      <- Channel Path Ids  ->  Control
Device-No  Subch.-ID  Type         1  2  3  4  5  6  7  8  Unit      Status
0061       0002       9042        4C  .  .  .  .  .  .  .  2032-04   Online
0062       0003       9042        4E  .  .  .  .  .  .  .  2032-04   Online
08E0-08EF  ....-....  3270-2       .  .  .  .  .  .  .  .  ....      Offline
1A20-1A25  0246-024B  3390-3 (E)  52 56 5A 5E  .  .  .  .  2105-E8   Online
3020-302B  0880-088B  OSA         0C  .  .  .  .  .  .  .  1731-01   Online
302F       088C       OSA         0C  .  .  .  .  .  .  .  1731-01   Online
...
CE31-CE3D  3D94-3DA0  3390-3 (E)  52 56 5A 5E  .  .  .  .  2105-E8   Online
F200       4D6B       3270-3      14  .  .  .  .  .  .  .  3174-ID   Online
```

Now that we know what channel paths are being used, we can evaluate the capacity of the channels. Figure 10-7 shows that the CHPIDs are not busy, so there is extra bandwidth for the paging devices.


IBM
Performance Toolkit for VM
FCX161 Channel Load and Channel Busy Distribution (VMLINUX7)
See also CHANNEL and FCHANNEL displays
Command
Refresh
Menu
Forw
Back
Help
Auto-Refresh

```
Interval 00:00:01-18:56:01, on 2011/09/29  (AVERAGE period, select current or interim data)

CHPID Chan-Group       <%Busy>  <----- Channel %Busy Distribution 00:00:01-18:56:01 ------>
(Hex) Descr Qual Shrd  Cur Ave  0-10 11-20 21-30 31-40 41-50 51-60 61-70 71-80 81-90 91-100
 4A   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 4B   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 4C   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 4D   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 4E   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 4F   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 51   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 52   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 53   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 54   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 56   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 57   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 58   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 59   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 5A   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 5B   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 5C   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 5D   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 5E   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
 5F   FICON  00 Yes     0   0   100     0     0     0     0     0     0     0     0     0
```

Figure 10-7 FCX 161 Channel Load and Channel Busy Distribution

Main storage can range from a few to tens of GB, but it is important that the paging subsystem can withstand the heavy peak loads and prevent high paging rates to devices.

10.4 Placing recommendations for disks

In this section, we recommend how you should select various disk devices for optimal performance. The latest high-end DS8000 series storage servers provide tools that place data across the appropriate drive tiers for the best performance.

10.4.1 Recommendations for DASD

In general, use more parallelism when possible to optimize I/O performance. You can also use more disks and reduce contention points.

Using more disks

I/O operations to a single physical disk are serialized. Parallel simultaneous I/O operations are faster than I/O operations to a single disk.

Reducing contention points

Contention for I/O resources can occur in the I/O path, as noted here:

- DASD contention

  Multiple virtual machines can experience DASD contention when simultaneously accessing data on the same DASD device.

- Control unit contention

  This can occur when two or more DASD devices share a common control unit.

- Channel contention

  This can occur during simultaneous access of the control unit or DASD devices sharing the same channel.

10.4.2 Recommendations for SCSI

Be sure to keep SCSI disks equally distributed over FCP adapters and World Wide Port Numbers (WWPNs). As with DASD, reduce contention points and use disks physically located in different ranks of the storage server. Do not use consecutive disk addresses, but alternate the clusters, logical control units, WWPNs, device adapters, and ranks.

10.4.3 Recommendations for z/VM minidisks

The usual placement considerations apply when you use minidisk allocations for a Linux guest, just as they would to a non-Linux disk. Aim to stripe the DASD allocation for the system to engage as many channels, chpids, and hardware-level caching as possible. Then spread the minidisks to ensure that there is a balance of overall I/O rates to all of the DASD volumes allocated. Do not place critical disks on the same volume, and ensure that z/VM paging, spooling, system residency volumes, online directory areas, and Linux swap disks are not on DASD volumes shared by important, performance-sensitive Linux application minidisks. These system level infrastructure volumes are best placed apart from Linux and other application volumes, and they must not share volumes with each other.

z/VM minidisk cache

Minidisk cache (MDC) is a data in memory (DIM) write-through caching technique that can, depending on the applications and usage patterns, cut the amount of real I/O on DASD. You can diminish DASD I/O wherever sufficient main storage and expanded storage memory resources for MDC are available. Note that using expanded storage for MDC is far less effective than using main storage.

There might be a trade off with increased paging, if the amounts of memory devoted to minidisk cache cause significant memory constraint. For a Linux minidisk to benefit from MDC, you need to understand the usage patterns, principally the number of reads and writes being performed. There is processor overhead if you update MDC when read operations are carried out. Unless the Linux minidisk is mainly read only, or has a large majority of reads compared to writes, consider running with MDC off for that disk, as the potential benefit from saving real I/O might be wiped out by the processor overhead. In any event, do not use MDC for Linux swap disks.

10.5 Hardware performance capabilities and features

Fibre Connection (FICON) provides I/O connectivity based on fiber optic technology. FICON is available in various versions that offer different bandwidth. SCSI uses connections via switch and uses the Fibre Channel Protocol (FCP).

10.5.1 Comparing ECKD DASD and SCSI to FCP

Traditional System z I/O uses the channel subsystem and issues CCWs to access ECKD, CKD, or FBA disks. In Linux, this requires a transformation from the block-oriented block device layer to CCW chains. The ECKD protocol does not use the count and key functions. The channel subsystem transfers the CCWs to the storage subsystem. The storage subsystem works with standard SCSI disks, and so the channel programs have to convert back to fix block format. Figure 10-8 illustrates disk access translation.

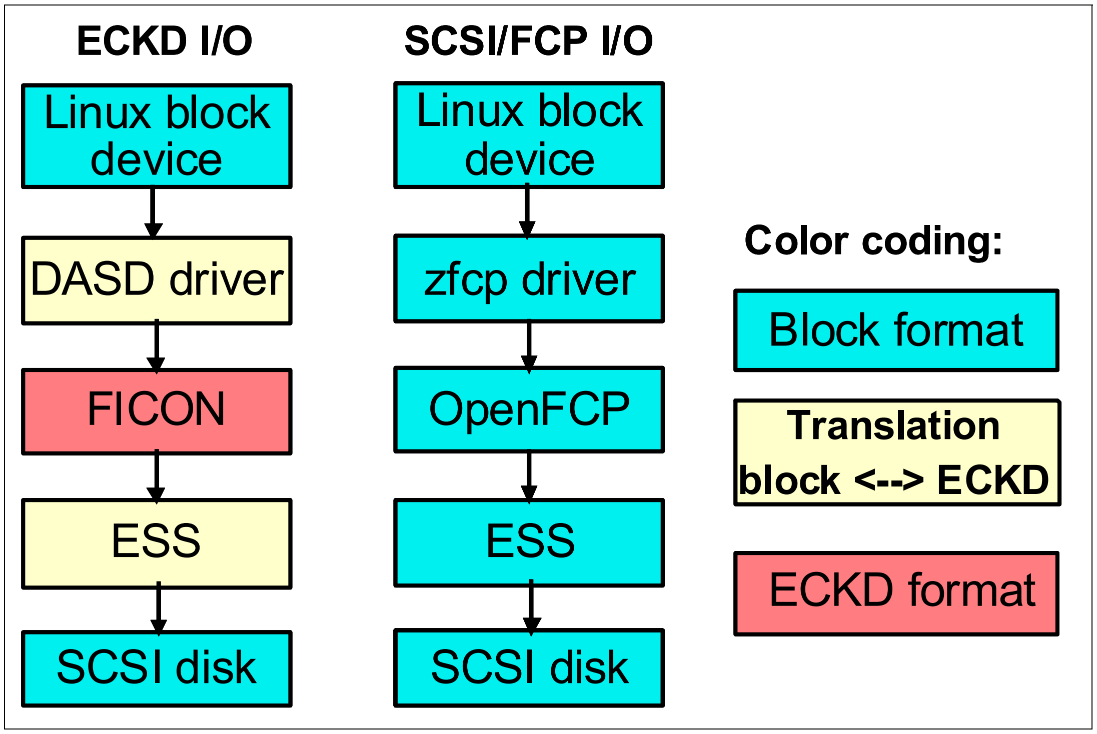


Figure 10-8 Disk access translation

10.5.2 Generations of storage subsystems

The biggest changes in disk I/O performance are observed when migrating between generations of storage subsystems. The caches usually are bigger, and the performance of the storage subsystem increases tremendously with faster processors and host adapters, more and faster caches, and optimized cache algorithms.

However, with regards to I/O performance, remember that shorter I/O wait times have a side effect. The overall throughput goes up and latency diminishes in the system. While the processor cycles per throughput unit tentatively go down (less context switches accompanied by more cache hits) the actual processor resource requirements tend to go up. More work done in a shorter time needs more processor power available.

10.5.3 Influence of DASD block size

The dasdfmt tool can format DASD devices with different hard block sizes. The best results in IOZone benchmark throughput and free disk space are seen when the dasdfmt formats DASD to a 4096 bytes block size. We tested this statement further and found that it does not depend on the request size issued by the application. Even small application request sizes (for example, 512 bytes) showed the best throughput with the 4096-byte dasdfmt block size.

10.5.4 IBM storage subsystems caching mode

Caching modes, such as DS8000 and DS6000™, are strategies used by the storage subsystem to optimize its cache hit rate.

DS8000 and DS6000 caching modes

Self-learning cache algorithms are performance enhancers. The DS8000 series caching technology improves cache efficiency and enhances cache hit ratios, using the algorithm Sequential Prefetching in Adaptive Replacement Cache (SARC).

SARC provides this:

- Sophisticated, patented algorithms to determine what data should be stored in cache, based on the recent access and frequency needs of the hosts.
- Pre-fetching anticipates data prior to a host request and loads it into cache.
- Self-learning algorithms to adaptively and dynamically learn what data should be stored in cache based upon the frequency needs of the hosts.

For more details, see *DS8000 Performance Monitoring and Tuning*, SG24-7146.

Tunedasd command

Use the Linux **tunedasd** command to switch between subsystem operating modes. **tunedasd** works with DASD and not with SCSI disks. These are command elements are listed below (Example 10-3):

- tunedasd -g or --get_cache

  Get the current storage server caching behavior.
- tunedasd -c or --cache <behavior>

  Set the storage server caching behavior.

Example 10-3 Set storage subsystem caching mode from normal to sequential

```
# tunedasd -g /dev/dasdc
normal (2 cyl)

# tunedasd -c sequential -n 2 /dev/dasdc
Setting cache mode for device </dev/dasdc>...
Done.

# tunedasd -g /dev/dasdc
sequential (2 cyl)
```

Note: You need to know which access should be used before specifying a cache mode. If you do not know the access, use the self-adapting and optimizing modes.

10.5.5 Linux Logical Volume Manager (LVM)

Linux Logical Volume Manager provides a higher level view of the disk storage on a computer system than the traditional view of disks and partitions. The main reason to use linear logical volumes is that they can be extended at any time without moving existing data to another disk.

The more sophisticated construct is a striped logical volume. It is mainly used to increase throughput. For sequential access it splits large requests, which would normally all go to one disk, into several smaller requests for multiple disks. For random I/O, LVM spreads I/O requests to multiple disks. In both cases striped logical volumes allow parallel I/O to all disks from the respective volume group. For FCP/SCSI disks, the striped Linux logical volume acts as a type of load balancer if the channels to the SCSI disks are assigned round robin. LVM2 striped logical volumes are also extendable. Figure 10-9 illustrates how LVM can increase I/O performance.

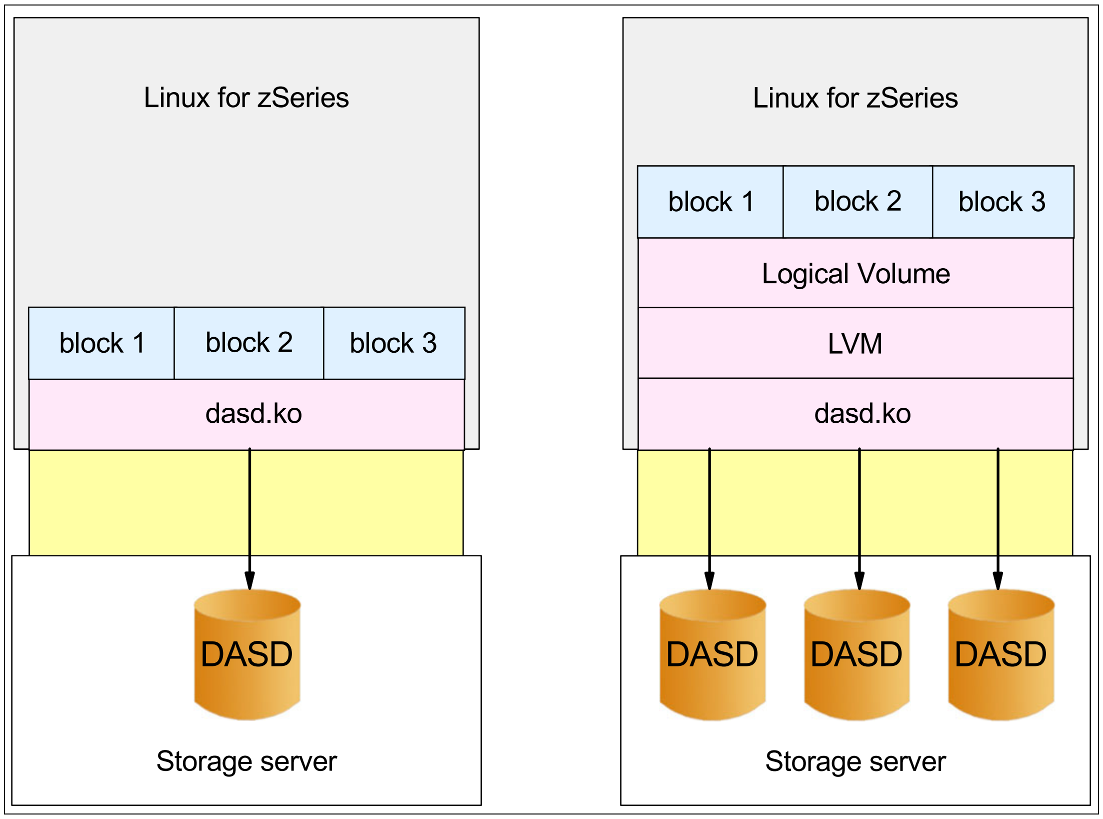


Figure 10-9 Using LVM for parallel DASD access

FICON and FCP

Use at least as many stripes as paths available, or the number of stripes that ensures an optimal path usage and used disks. The recommended strip sizes are 32 kb and 64 kb.

You need to employ the multi-path user space tool to use LVM striping. The multi-path user tool consumes processor cycles. This increases processor consumption per transferred unit of data. The more disks that are involved, the higher the additional consumption. With striping, contention can occur at the control or disk unit. Make sure that the disks are distributed to different control units. It is important to place dedicated disks and minidisks when you use LVM, as described in 10.4, “Placing recommendations for disks” on page 179.

Multipathing - Failover mode versus multibus mode with SCSI disks

Multipath I/O is a method that connects a system and a mass storage device using multiple paths through buses, controllers, switches, and other connection devices. Multipathing increases system stability and resilience. There are two main modes of multipathing, multibus, and failover:

- Multibus mode
  - Good for load balancing.
  - Overcomes bandwidth limitations when one path is used.
  - Uses all paths in a priority group in round-robin technique.
  - The path switches after a configured amount of I/Os is completed.
    - See `rr_min_ioin multipath.conf`.
  - Method for doing disk I/O in parallel to increase throughput
    - Is needed with static PAV DASD devices in Linux distributions before SLES11 and RHEL6
    - Is needed with SCSI LUNs with any Linux distribution
- Failover mode
  - A high-availability option.
  - The volume remains accessible via an alternate path in case of a single failure in the hardware connection.
  - Fails one path, another path is used to route I/O (transparent to the application).
  - Does not increase throughput.

Learn how to use multipathing at this website:

http://www.ibm.com/developerworks/linux/linux390/perf/index.html

10.5.6 Parallel Access Volume (PAV)

IBM storage servers contain concurrent operation capabilities that allow the system to access volumes in parallel. This means that the system can concurrently transfer data to or from the same volume from the same system or system image. PAV must be installed on the storage side.

PAV allows the system to support simultaneous access to the logical volume by multiple users or processes. Read operations can be accessed simultaneously, but write operation can be accessed in parallel only to different domains. Writes to the same domain still have to be serialized to maintain data integrity. The domain of an I/O operation is the specified extents to which the I/O operation applies.

From the Linux point of view, all base and alias devices are normal block devices. The Linux DASD driver supports PAV by creating a UID attribute in sysfs that identifies the base device and alias devices that belong together. You need to set up a multi-path user space to combine these devices and realize the intended parallel access effect.

With PAV, storage systems can present the same physical disk volume as a base device and one or more alias devices. You can assign separate device numbers on System z for the base device and the aliases. Figure 10-10 illustrates the relationship between the base and alias devices for a volume.

A disk volume can be accessed as a base device or one of two aliases. The base represents the real DASD volume space, and the aliases are shadows of the base. The aliases have no owned data space, but are mapped to the same physical volume as that accessed by the base. Figure 10-10 shows these relationships.

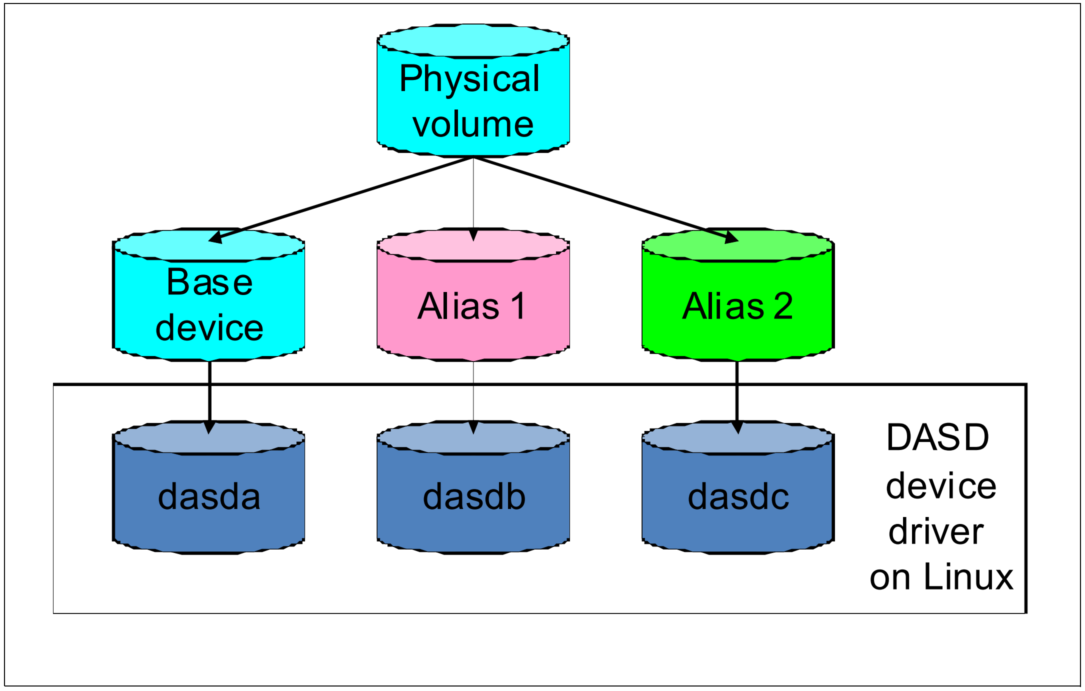


Figure 10-10 Relationship between the base and alias devices for a physical volume

If a Linux system has access to a base device and its aliases, the DASD driver senses the base device and each alias as a unique and independent DASD, and assigns a unique device name to each of them (Figure 10-10 on page 185). After DASD is partitioned and all the base devices and aliases are newly initialized, the DASD driver senses the partitions and assigns additional device names for each of the partitions on both the base device and the aliases (Figure 10-11).

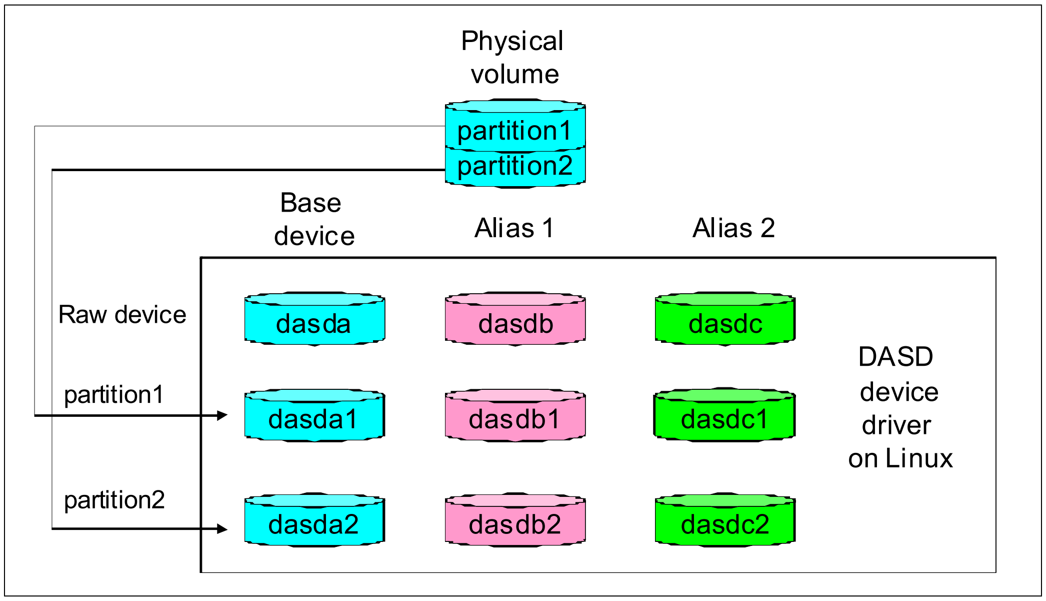


Figure 10-11 Relationship between the base and alias devices after partitioning

Using PAV as DASD and minidisks

You can use PAV volumes as both dedicated DASDs and minidisks, depending on who will be responsible for optimizing I/O performance. If you use PAV as DASDs, the performance benefits that you gain from PAV depends on Linux management. If you use PAV as minidisks, you need to assign z/VM the responsibility of optimizing the I/O performance for volumes used by multiple guests.

When PAV devices are used as dedicated DASD, Linux will manage their use after the base and associated aliases are attached to the Linux guest. You cannot dedicate a real volume to multiple guests. Any PAV benefits that you realize in a dedicated environment will depend entirely on the Linux guest. Figure 10-12 illustrates a Linux guest with two dedicated physical volumes, each of which has a base device and two alias devices, E200 to E203.

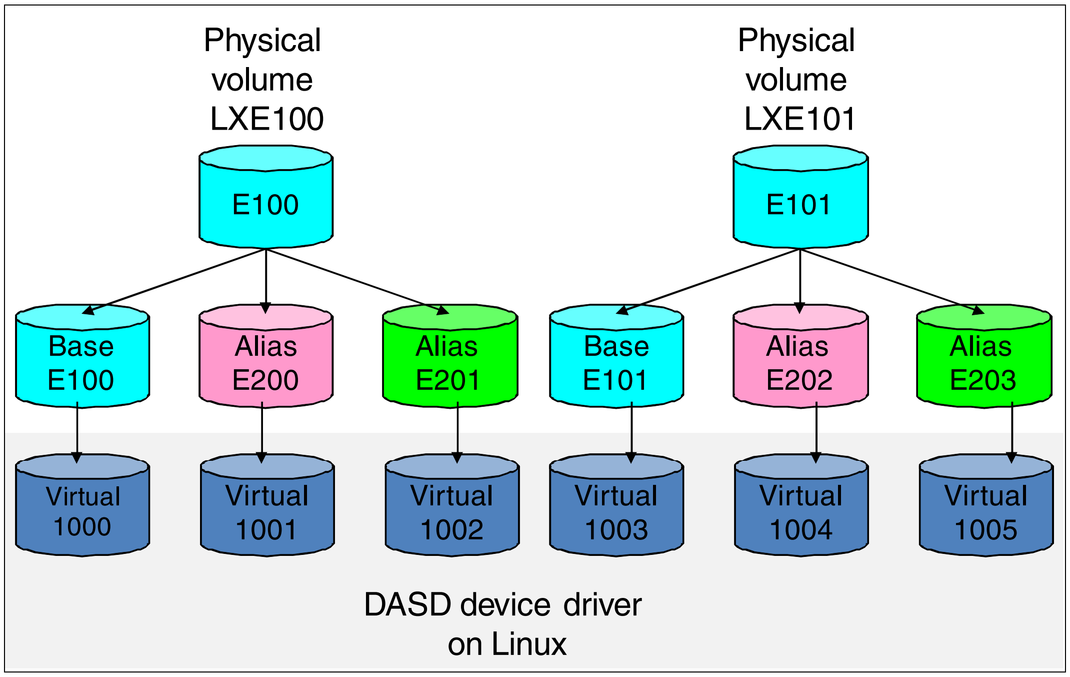


Figure 10-12 The relationship between physical volumes and virtual volumes seen from a Linux guest

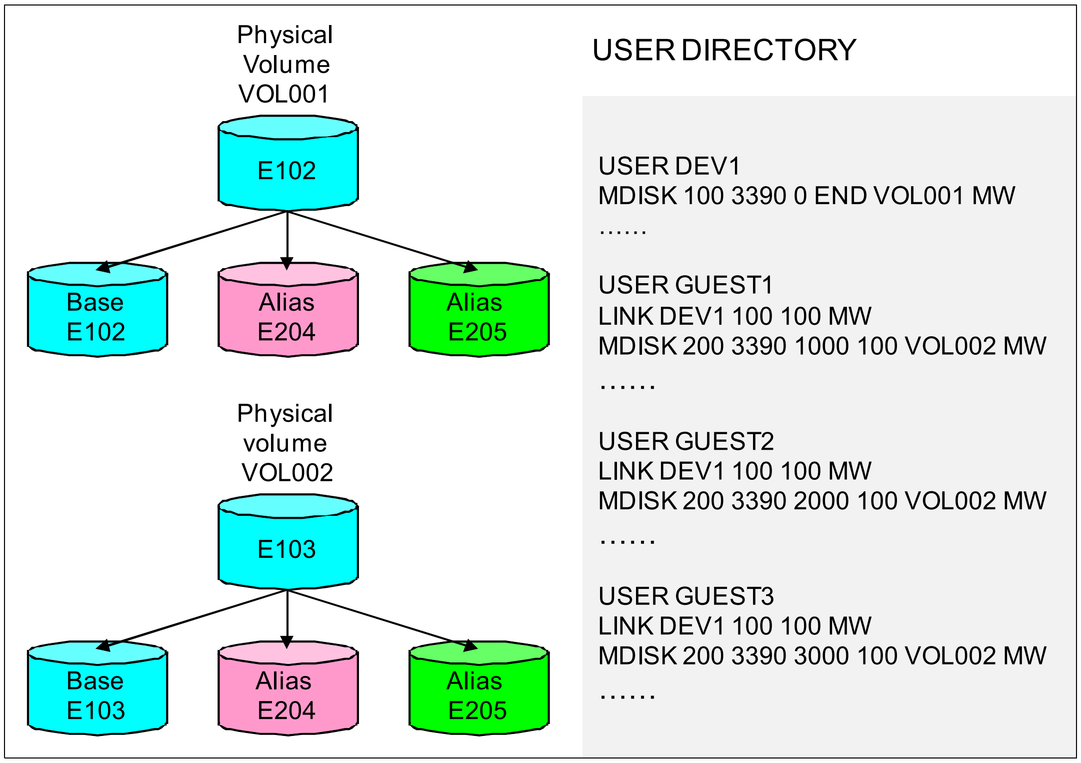
Example 10-4 shows the Linux guest entry in the z/VM USER DIRECTORY. Physical volumes E100 and E101, each with two aliases, are dedicated to Linux guest LINUX01. E200 and E201 are aliases to E100. E202 and E203 are aliases to E101. z/VM dedicates the two real volumes as six devices, as virtual devices 1000 to 1005, to the Linux guest, where DASD senses them as unique and independent DASDs. The Linux DASD driver manages and optimizes these PAV volumes to improve performance.

Example 10-4 Linux guest entry in the z/VM USER DIRECTORY

```
USER LINUX01 LNX4ME 2G 2G G
  INCLUDE LNXDFLT
  DEDICATE 1000 E100
  DEDICATE 1001 E200
  DEDICATE 1002 E201
  DEDICATE 1003 E101
  DEDICATE 1004 E202
  DEDICATE 1005 E203
```

PAV volumes similarly support both full-pack and non-full-pack minidisks. A guest virtual machine can define one or more minidisks on the same physical volumes. The physical PAV volumes have a real base device and one or more real aliases. Each minidisk on the physical volume has one virtual base device and zero or more virtual aliases. The number of virtual aliases for a guest should not exceed the number of the real aliases defined in the hardware for the real volume.

z/VM optimizes all I/O operations on minidisks through virtual base devices or virtual aliases. z/VM automatically selects an appropriate subchannel from the real base device or real aliases. I/O performance can be gained only when full-pack minidisks are shared among guests using LINK statements, or when multiple non full-pack minidisks are on a real PAV volume. Figure 10-13 illustrates using PAV volumes as minidisks. There are two physical volumes, VOL001 and VOL002, each of which has a real base device and two real aliases. In the z/VM user directory, user DEV1 is defined with a full pack minidisk 100 on volume VOL001, which is linked by three other guests. These three guests define a 100-cylinder minidisk on volume VOL002 separately. You can enhance I/O performance by sharing minidisks on a real PAV volume in these two ways.



```
USER DEV1
MDISK 100 3390 0 END VOL001 MW
......

USER GUEST1
LINK DEV1 100 100 MW
MDISK 200 3390 1000 100 VOL002 MW
......

USER GUEST2
LINK DEV1 100 100 MW
MDISK 200 3390 2000 100 VOL002 MW
......

USER GUEST3
LINK DEV1 100 100 MW
MDISK 200 3390 3000 100 VOL002 MW
......
```

Figure 10-13 Using PAV volumes as minidisks

Linux can be one of the guests that uses PAV as minidisks while depending on z/VM to optimize I/O performance. If you are using other operating system guests, including z/VM v4.4, v5.1, v5.2 without PAV support, VSE, TPF, and CMS, you should rely on the underlying z/VM for PAV performance. z/OS guests can be configured to exploit PAV as well as Linux.

PAV performance

Generally, disk performance depends on the time required to perform a single I/O operation on a single disk volume in z/VM. Here time refers to *response time*, which is composed of queue time and service time. Queue time is the period that a guest I/O waits for the real volume. Service time is the period required to finish I/O operation. PAV can increase the access paths to a real volume by using the base device and multiple alias devices. This reduces queue time. Adding aliases does not enhance volume performance if queuing does not occur because the time required to perform an I/O operation is not appreciably changed by queue time. In some cases, increasing the number of aliases for a volume results in increased service time for that volume. Usually the decrease in wait time outweighs the increase in service time, resulting in improved response time.

The z/VM Performance Toolkit provides a report about the response time and related performance statistics for the real devices. Use the **device** command to generate the report (Example 10-5).

Example 10-5 DEVICE report from performance toolkit

```
FCX108      CPU 2094   SER 2991E   Interval 16:19:45 - 16:20:45    Perf. Monitor
____ .                    .      .     .    .    .    .    .    .    .    .
<-- Device Descr. -->  Mdisk Pa- <-Rate/s-> <------- Time (msec) -------> Req.
Addr Type    Label/ID   Links ths  I/O Avoid Pend Disc Conn Serv Resp CUWt Qued
C703 3390    LXC703         1   4 60.1  36.3   .4  5.7  2.3  8.4  8.4   .0   .0
C704 3390    LXC704         1   4  1.2    .0   .3  3.0  1.5  4.8  4.8   .0   .0
C705 3390    LXC705         1   4   .5    .0   .3  1.4   .7  2.4  2.4   .0   .0
C706 3390    LXC706         1   4   .4    .0   .3  1.7   .6  2.6  2.6   .0   .0
```

The device reports display these columns:

- I/O is the I/O rate for the device, representing operations per second.
- Serv is service time, or average time, in milliseconds, that the device uses performing a single I/O.
- Resp is the response time, or the average time, in milliseconds, that the guest machine perceives is required to perform an I/O on the minidisk. Response time includes service time plus wait time (that is, queue time).
- Req.Qued is the average wait queue for the real device, or the average number of I/Os waiting to use the volume.

When you see from the Req.Qued value that a volume is experiencing queuing, add aliases to that volume until you run out of alias capability, or until the queue is empty. When tuning PAV, we suggest that you estimate the queue depth for the number of aliases and add aliases until the volume I/O rate maximizes or the volume wait queue is empty, whichever comes first.

Note: Additional PAV improvements are included in SLES11 SP1. HyperPAV support allows the control of the base device and its aliases to be bundled in the DASD driver. The multipath user space tool added processor costs in previous Linux distributions and is no longer used.

See the following links for more information:

- z/VM PAV support

  http://www.vm.ibm.com/storman/pav/pav2.html
- Performance report

  http://www.vm.ibm.com/perf/reports/zvm/html/index.html
- Linux on System z - How to Improve Performance with PAV

  http://public.dhe.ibm.com/software/dw/linux390/docu/l26dhp00.pdf

10.5.7 HyperPAV

PAV uses a static set of one base and one or more alias addresses, as described in “PAV performance” on page 188. HyperPAV is a new feature available on current storage servers. HyperPAV is supported by the latest Linux distribution updates SLES11 SP1 and RHEL6.1. HyperPAV and PAV provide similar features, but HyperPAV offers these advantages:

- Fewer aliases need to be defined.

  Base and alias devices remain in a HyperPAV environment, but one or more alias no longer need to be defined as belonging to a special base device. Instead, aliases are defined as belonging to the Logical Subsystem (LSS), also known as Logical Control Unit (LCU). If the base device is busy, one or more alias devices are assigned dynamically to this base. When the I/O process completes, the aliases are released and are again available in the pool.

- Multi-path user space setup is no longer needed.

  Base devices and alias devices have to be defined on the storage server and in the IOCDS. The DASD channel program is able to determine the base device that is presented to Linux like any regular block device.

  Alias devices must be selected quickly per request, so this algorithm is implemented in the DASD device driver, allowing the system to quickly adapt to required I/O throughput changes.

  The DASD sysfs attribute `uid` is not required but helps to identify base and alias devices. Because as many as needed aliases are used out of the alias pool and there is no fixed base, the unit address is part of the uid, as xx (Example 10-6).

Example 10-6   The uid, where xx indicates the unit address

```
IBM.7500000106.71.1400.xx
```

Note: HyperPAV allows the multipath setup provided originally for PAV. The change from PAV to hyperPAV and vice versa is possible even during operation.

The Linux on System z performance team tested HyperPAV with promising results which are not yet published. There appears to be no advantage for HyperPAV over PAV in terms of performance. However, HyperPAV is much easier to handle in terms of administration. HyperPAV aliases must be visible in the Linux guest, and must be put online via tools such as **`chccwdev`** or kernel configuration.

We recommend HyperPAV over PAV because the former is more flexible and easier to manage. HyperPAV assigns more resources to I/O request that need the extra performance and is a good choice, especially if new disks have to be connected to systems.

For more details see the Linux on System z - How to Improve Performance with PAV (HyperPAV is included):

http://public.dhe.ibm.com/software/dw/linux390/docu/l26dhp00.pdf

10.5.8 High Performance FICON

High Performance FICON (HPF) provides a simpler protocol to FICON. With HPF, the DASD driver uses the Transport Control Word (TCW), which sends multiple channel commands to the control unit as a single entity instead of a stream of individual commands.

Prerequisites

The prerequisites are:

- HPF is activated in Linux by default but can be turned off using a command-line parameter.
- HPF must be supported by the storage server (prized feature). Otherwise the DASD driver automatically reverts to using channel command words (CCWs).
- System z10 GA2 or later (non-chargeable feature).
- FICON Express2 or later.
- IBM DS8000 Licensed Internal Code (LIC) bundle Version 64.1.1.16.0 (release 4.1) or later (priced feature).
- Linux with DASD driver with *read/write track data* support, and HPF support, using I/O subsystem transport mode to access the storage server. These are included in SLES11 SP1 and RHEL6.1 and later versions.

HPF showed advantages in throughput and cost, especially when many parallel processes are accessing the DASD. The implementation in the DASD driver combines HPF with the read/write track data option. The zHPF architecture offers you reliability, availability, and serviceability (RAS) benefits, due partly because the control units offer more information when errors occur.

The Linux on System z performance team created a presentation with interesting information and measurement results. Access it using this link:

http://public.dhe.ibm.com/software/dw/linux390/perf/High_Performance_FICON.pdf

10.5.9 Storage Pool Striping (SPS)

SPS is included in DS8000 storage subsystems, made available with Licensed Machine Code 5.3.x.x. In a DS8000, an array consists of one array site of disks. Some ranks are grouped in the logical construct of an extent pool. In an extent pool, all disks have the same type (CKD or FBA) and the same disk rpm characteristics and RAID type (that is, the extents in the pool are homogenous).

Logical volumes (LUN or volumes) are carved out of the extent pool. Previously, this was done only following the Rotated Volume allocation method. Extents are allocated sequentially, meaning that all extents are taken from the same rank until it is full, then from the next rank, and so on. Storage Pool Striped allocation gives us a second method where the extents of a logical volume (LUN or volumes) are striped over different, or at least two, ranks. The SPS method offers both advantages and disadvantages, as discussed in the following sections.

SPS advantages

These are the advantages of SPS:

- Storage pool striping significantly improves performance by not using physical disks from one extent pool and by distributing all the I/O to eight disks in this rank. Using disks from many extent pools provides for more paths and NVS of the DS8000 infrastructure in parallel, which can greatly improve performance.
- Storage pool striping requires an administrative effort that can be compared to the effort of defining large disks. SPS is transparent to the user and is more convenient than using striped LVM from Linux. Also, SPS eases subsequent disk administration because

intelligent disk placing uses as much physical resources as possible in parallel during allocation. Without SPS, the storage administrator has to find subsequent disks from different ranks in the DS8000, which can later be combined in a striped LVM.

- SPS moves more overhead from the mainframe to the storage server than a striped LVM in Linux. This is an advantage in terms software licence fees.
- With SPS, the single maximum I/O request is delimited by the device driver (for example, 512 kB), whereas striped LVM is delimited by the chosen stripe size (for example, 64 kB).

SPS challenges

These are the challenges of SPS:

- If you lose one rank, you risk losing all the data for the entire extent because striped over all ranks belongs to this extent. RAID 6 support, available with Licensed Machine Code 5.4.x.x, can help resolve this issue. We recommend that you not use more than four to eight ranks in an extent pool.
- Storage Pool Striping and Rotated Volume allocation might not combine easily. SPS allocates in a balanced fashion in the extent pools, but Rotated Volume does not. Using both methods together might result in certain ranks filled earlier than others, which leaves space unusable for SPS. We recommend that you use SPS and Rotated Volume allocation separately in different extent pools.
- The extent pools cannot span servers in the storage server (there are two server sides in DS8000.), so you cannot use two processor sides in the storage subsystem and all of the cache with one single extent pool. You can use two server sides at the same time with tools such as LVM on Linux. You can also combine Storage Pool Striping and LVM with striping, but this is not recommended, as it adds overhead in two places. This limitation is less important in a production environment with disk access in parallel and using both server sides in parallel.
- The SPS stripe size is relatively big (for example, 1 GB with FBA), which means that access to relatively short records in a randomized pattern will profit more than access to data sequential and long records. Compare this to the smaller LVM preferred stripe sizes of 32 kB or 64 kB.

The Linux on System z team measurements are available at this link:

`http://public.dhe.ibm.com/software/dw/linux390/perf/Storage_Pool_Striping.pdf`

10.5.10 Linux I/O schedulers

The purpose of I/O schedulers is to optimize disk access using features that consider disk layouts, maximize request sizes, and prioritize certain requests or balance I/Os over all attached disk devices. We do not recommend that you use algorithms or calculations where a disk geometry is assumed if you are using Linux servers connected to a storage server.

We also do not recommend issuing each request directly to the disk, as that can result in a high number of small requests, causing poor throughput. In the worst case in this scenario, each request would be sent to a completely different disk area. Current data transfer rates from physical disks are very high, so transfer times are very short. The greatest amount of access time is caused by disk latency, mainly caused by disk head movements of about 8 ms on average. You need to minimize the number of I/O operations and disk head movements to maintain system optimization. These are examples of Linux I/O schedulers:

- Noop scheduler

  The noop scheduler economizes requests by merging many small contiguous requests into fewer, larger requests.

- Deadline scheduler

  The deadline scheduler implements request merging and prevents request starvation that can be caused by the elevator algorithm. The scheduler introduces a deadline for each request, though it prefers read requests. Linux does not delay write processes until their data is actually written to the disk because it caches the data, while readers have to wait until their data is available.

- Anticipatory scheduler

  The anticipatory scheduler implements request merging, and optimizes for physical disks by avoiding head movements. It solves situations where many write requests are interrupted by a few read requests. After a read operation, it waits for a certain time for another read, and does not switch back immediately to write requests. Do not use the Anticipatory scheduler for storage servers.

- Complete fair queuing scheduler (cfq scheduler)

  The complete fair queuing scheduler implements request merging, and uses the strategy of allowing all users of a particular drive the same number of I/O requests over a given time.

Note: The distribution default is the deadline scheduler. Do not change the default to anticipatory because performance degrades if a storage server is in the environment.

To check which scheduler is currently operating on your system, search the **dmesg** command for the term `scheduler`. Example 10-7 displays the results, where the deadline scheduler is operating, while noop, anticipatory, and cfq are also registered to the Linux system.

Example 10-7 Check for the scheduler supported and in use

```
lnxsu3:~ # dmesg | grep scheduler
io scheduler noop registered
io scheduler anticipatory registered
io scheduler deadline registered (default)
io scheduler cfq registered
lnxsu3:~ #
```

Select the I/O scheduler using the boot parameter elevator in `zipl.conf` (Example 10-8), where elevator:= as | deadline | cfq | noop.

Example 10-8 /etc/zipl/conf with elevator parameter

```
...
[ipl]
    target = /boot/zipl
    image = /boot/image
    ramdisk = /boot/initrd
    parameters = "maxcpus=8 dasd=5000 root=/dev/dasda1 elevator=deadline"
```

Note: Change a scheduler with caution, as the wrong choice will reduce system performance. We suggest that you begin with the default scheduler, which performed well in our experiments.

For more details, see:

`/usr/src/linux/Documentation/kernel-parameters.txt`

10.5.11 Direct and asynchronous I/O

Disk I/O throughput and cost depend on whether you use direct I/O or a file system. With direct I/O, the application must take over functions completed by a file system. Direct I/O does not use Linux page cache and therefore saves double buffering of the transferred records. This also allows smaller memory sizes for Linux, or offers you the option to increase shared memory, for example, for database buffer pools or Java heaps with the same total memory. You can also save CPU cycles if the data is not copied to and from the page cache. In addition, if you run short on page cache during heavy disk I/O, you do not spend time for page cache memory claiming.

Direct I/O does have drawbacks. For example, accessing the same files once via direct I/O and once via page cache will cause data corruption. Because the data records are not cached, the I/O schedulers cannot merge smaller requests into larger ones as well as they do when the data resides in page cache. The result is a smaller I/O request size on average.

Async I/O allows you to issue I/O requests without immediately waiting for them to be completed. Using async I/O, the application initiates the I/O request, and then does other work. Later, when it gets posted by the I/O completion event, it can continue the execution immediately after the issued I/O. Async I/O allows a process to wait on many completions simultaneously, therefore reducing the number of called I/O processes found in databases. Fewer I/O processes reduces memory requirements for active processes.

While testing OLTP on databases, we saw up to 50% throughput improvement with both features active versus both not set. Many current middleware products support async I/O. We recommend that you use both direct and asych I/O to improve throughput.

10.5.12 Linux file systems

Current versions of Linux include various journaling file systems, but because each has pros and cons, we cannot recommend a specific one. The performance depends on many parameters, such as the character of the files, the workloads, and the kind of access (sequential, random, or mixed). Journaling file systems have overhead compared to non-journal file systems, but are still strongly recommended because they provide less data loss and shorter recovery when errors occur. Examples of currently available file systems are ext3, ext4, and xfs. The xfs file system showed advantages in use cases with many write operations.

ext3 file systems are widely used because of their stability and because they were developed on top of ext2. ext3 offers a seamless migration from ext2 with the **`tune2fs -j <partition>`** command. The system creates a journal and mounts the file system as ext3. You need to consider many factors in choosing a file system (for example, applications and tools). Because performance varies, you might try different file systems with important and time-critical workloads to determine which one will work best for your system.

Using ext3 will help you migrate to ext4. After being available in a experimental version, ext4 is fully supported in SLES11 SP1 and RHEL6 ext4. OCFS2 is fully supported as an entity of the SUSE Linux Enterprise High Availability (HA) solution. ext4 documentation states that, from a performance perspective, it is better for you to define an ext4 file system from scratch than upgrade from ext3.

Temporary files are often a root cause for performance problems. Do not create them on a journaling file system, as it makes no sense to collect recovery data for them. We recommend that you do not access the storage server at all with temporary files. Instead, place them on a ram disk or in the `/tmpfs`, if your system size allows.

10.5.13 Read-ahead setup

Several components of the environment are candidates for performing read aheads. However, performance of the low-hit workloads with high I/O load suffers from read-ahead operations.These operations make sense if data will be read in sequential order, or if the data is large (for example, large huge database entries). For example:

- Applications
- LVM
- Block device layer
- Storage sub systems

Each components above issues read-ahead activities that can lead to more I/O activity than requested. All components assume that subsequent data units will be needed soon, as follows:

- The application might try to fill its cache with pages for a set of data with each database read access.
- For each page, the logical volume manager fetches a set of pages for subsequent file blocks. The default of 1024 pages is too high and results in a performance degradation for this workload. But after you change the default, you cannot set it again higher than 255 pages.
- The Linux block device layer can also add pages to each read request. For each file block, the storage sub-system gets a set of subsequent disk blocks, or several disks if the logical volume is striped.

Managing caches and caches thrashing are full of data that might not be needed. In our tests, the large number of physical disk read requests, together with the physical limitations in terms of seek and latency times, all limited database accesses speed. We suggest that you lower read-ahead default settings to keep the data transfer path free for the requested data.

11

Network considerations

This chapter discusses networking options for Linux on System z and z/VM based architectures. You need to identify and define these options before you try a solution because they influence overall performance. We examine guidelines for selecting network options, network configuration parameters, z/VM enhancements, hipersockets, and tuning recommendations.

11.1 Selecting network options

You can attach Linux on System z to an external network in several ways, but each option can affect system performance, so choose carefully. You can select physical or virtual networking. Linux uses the network directly in the physical option, and by services managed by z/VM in the virtual option.

11.2 Physical networking

You can install Linux on LPARs and as a z/VM guest. Your only option with Linux on LPAR is to directly attach to the network by connecting the operating system directly to the OSA adapter (Figure 11-1).

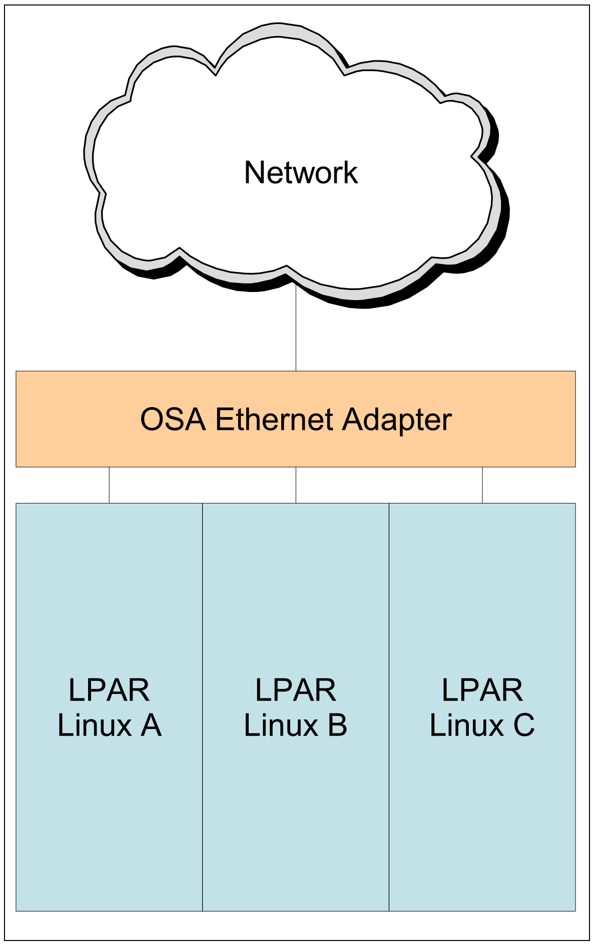


Figure 11-1 Directly attaching to network

Use the **dedicate** command on the `user direct` definition for the Linux guest to direct on z/VM. This allows Linux to directly control the network interface. Dedicate the network interface when you do not need to manage many images, or when you want to dedicate a network resource to one guest.

Using only one adapter can result in a single point of failure. Linux is not available if an OSA adapter fails. To avoid failure, duplicate the network connection and use a high-availability solution, such as a virtual IP address. This solution introduces protocols to manage dynamic routing, and results in higher availability, with some overhead from routing management products. Zebra and the ospf daemon are good options for Linux network management. For a high-availability solution for Linux on z/VM, we suggest a virtualized network managed by VSWITCH, as described in 11.3, “Virtual networking” on page 202.

Note that a Direct Attached OSA connection is faster than VSWITCH, especially if the number of connections used in parallel increases. Figure 11-2 shows transactions per second completed by the Direct Attached OSA connection as compared to VSWITCH. Notice that as the number of clients increases, more transactions can be completed using the Direct Attached OSA connection. The performance data was collected using a TCP request and response (RR) test. For more information about the tests used in this chapter, see 11.8, “Performance tools” on page 216.

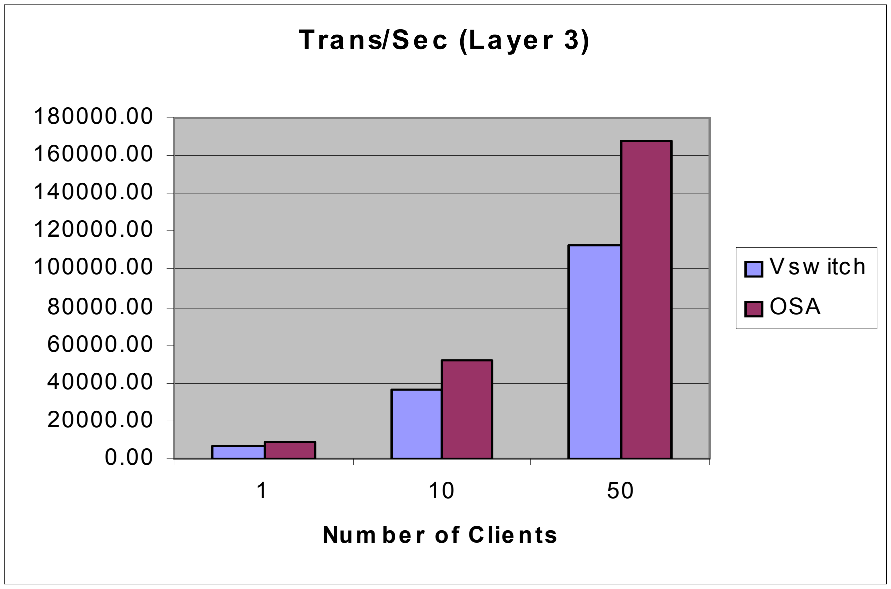


Figure 11-2 Trans/sec report from RR test

Figure 11-3 shows the processor usage for the same scenario. The CPU time per transaction will help you decide which option will give the best performance. The Virtual Switch (VSWITCH) uses less processor time with a smaller number of clients than the Direct Attached OSA connection. The Direct Attached OSA connection is a better option when the number of clients increases.

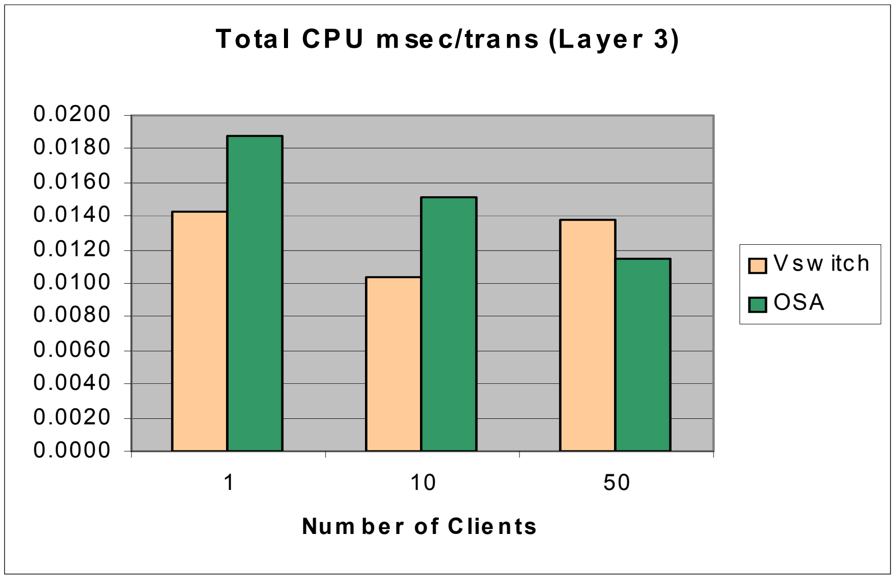


Figure 11-3 Total CPU msec report from RR test

The differences in throughput in streaming benchmark tests (STRG) are only significant when more connections are used in parallel. But VSWITCH and the Direct Attached OSA card differ in the cost per MB transferred with any number of connections used in parallel. Figure 11-4 and Figure 11-5 on page 201 provide data on the STRG transactional workload.

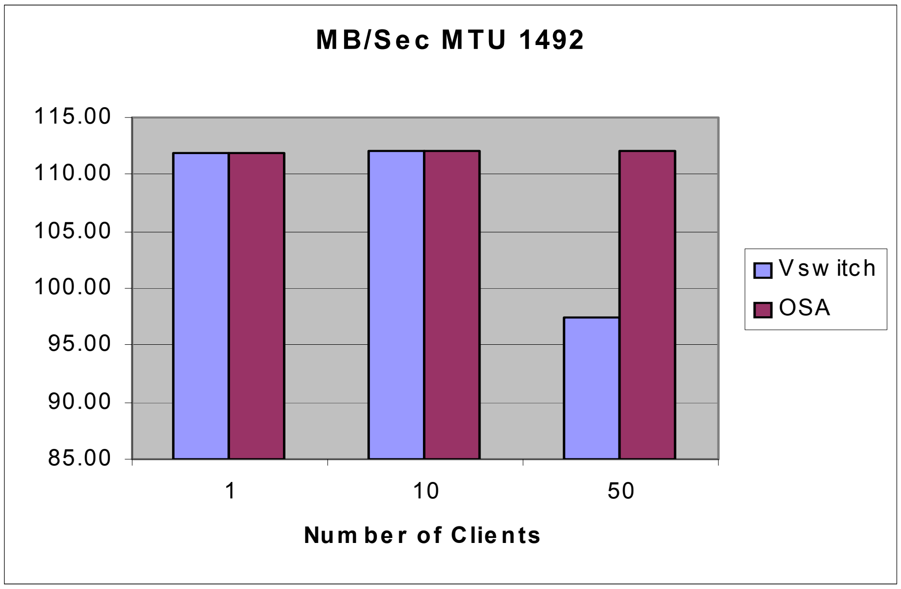


Figure 11-4 Throughput report for STRG test

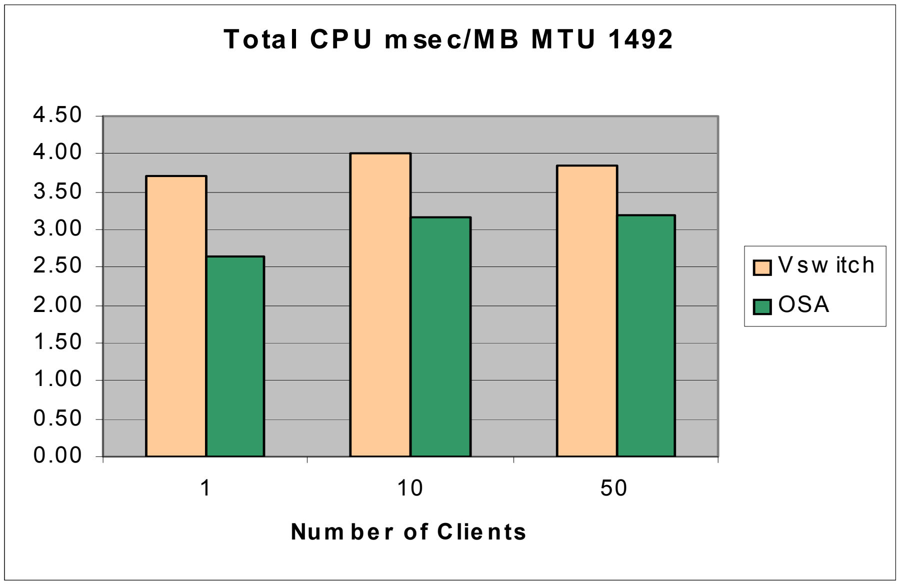


Figure 11-5 Transaction cost report from STRG test

For a more comprehensive comparison of network options, see the throughput and processor costs charts available here:

http://public.dhe.ibm.com/software/dw/linux390/perf/network_connection_types.pdf

11.3 Virtual networking

You can use virtual networking whenever z/VM controls Linux network accesses. Recent releases included improvements to consolidate network architecture and increase efficiency. VSWITCH was introduced to eliminate Linux images that route the inside network to the outside network or z/VM TCP/IP routing functions. You do not need to use other images to perform routing processes because the VSWITCH acts as a real Network Switch in z/VM. Linux routers introduce overhead, especially in a redundant scenario where two or more Linux routers are used to ensure high availability. Figure 11-6 shows options for attaching Linux images to the network.

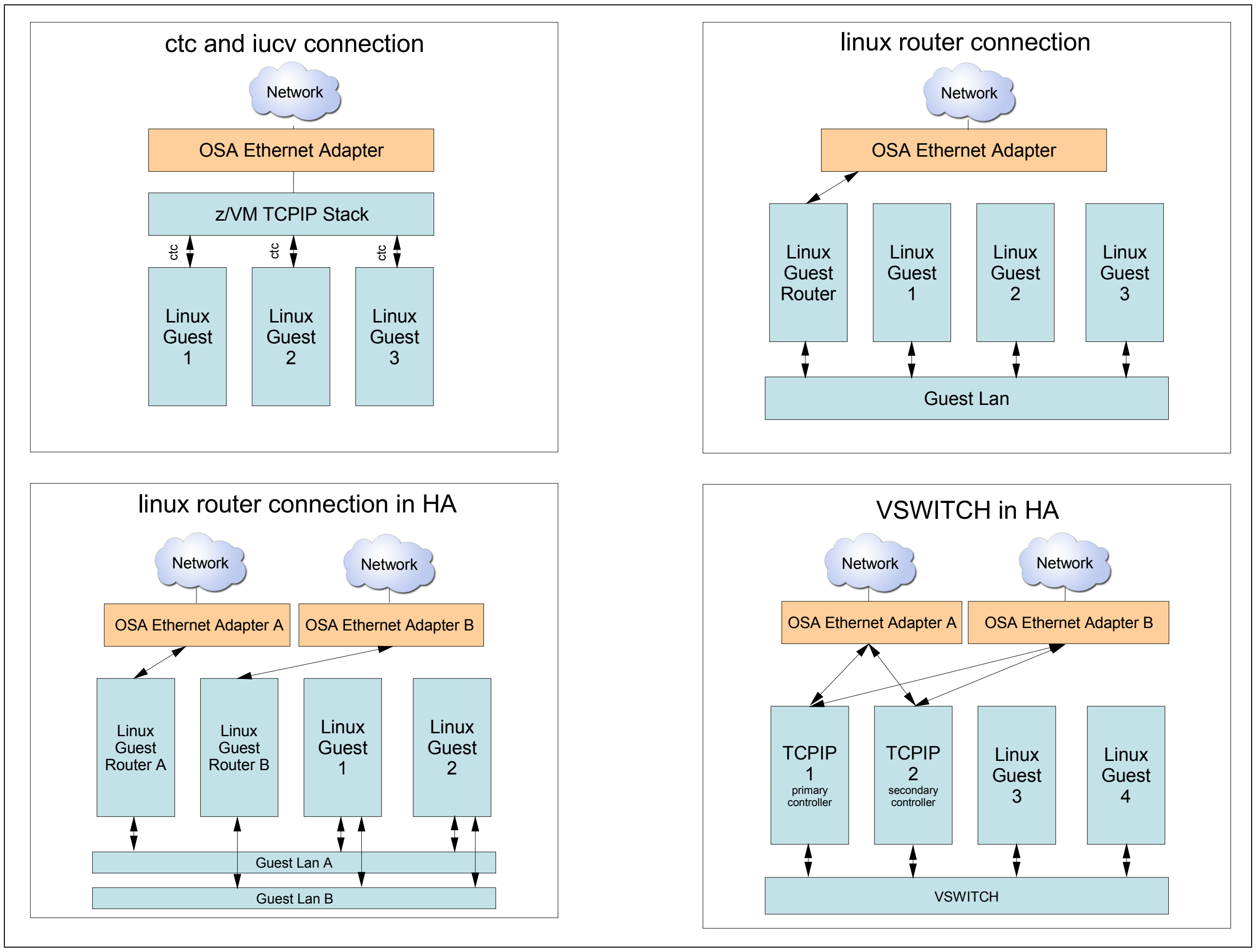


Figure 11-6 Virtual networking scenarios

In a CTC and IUCV connection, the TCP/IP virtual machine is responsible for routing traffic between the Linux images and the network. In this configuration, each Linux uses a point-to-point link, ctc or iucv, to TCP/IP virtual machine. This configuration is no longer used but is still supported. For performance reasons, we suggest that you migrate to another virtualized solution, as described next.

You can use Linux instead of the TCP/IP stack for routing, as shown in the Linux router connection in Figure 11-6 on page 202. In a server farm based on Linux and z/VM, you can use Linux as a router to an external network and link the other Linux images to it by using the default gateway statement. All Linux images can use the OSA driver connected to a Guest LAN managed by z/VM.

Figure 11-6 on page 202 shows the Linux router connection in HA. To use such a high-availability scenario, complete these steps:

1. Duplicate the guest LAN where Linux is attached.
2. Duplicate the Linux router.
3. Use one VIPA address for each Linux.
4. Implement a dynamic routing protocol, for example, ospf.

This solution offers a high-availability scenario but introduces features that can influence network and processor performance. Introducing another Linux means introducing another workload that needs to be managed. Ospf and all the associated daemons (for example, zebra) continuously poll the network to locate changes, which means that the operation wakes up the processor repeatedly during normal operations.

VSWITCH was introduced (for example, VSWITCH in HA) to avoid such problems. VSWITCH acts like a hardware network switch that bridges a local network (for example, a guest LAN) to an external network. VSWITCH offers a built-in redundancy because it can be configured to work with a primary OSA interface and, in case of failure, with a secondary OSA interface. z/VM 5.3 offers major enhancements with the Virtual Switch Link Aggregation support, as described in 11.5, "z/VM VSWITCH link aggregation" on page 212. Table 11-1 lists the advantages and disadvantages of various networking options.

Table 11-1 Performance comparison for networking options

Network types	**Availability**	**Performance**
Point-to-point with ctc drivers	Single point of failure on z/VM TCP/IP stack	Poor
Point-to-point with iucv drivers	Single point of failure on z/VM TCP/IP stack	Poor but better than CTC
Linux router	Single point of failure on Linux router	Faster than point-to-point
Redundant Linux routers	No single point of failure	Poor (depending on type and number of processors)
Virtual Switch (VSWITCH)	Single point of failure on TCP/IP controller	Good
Redundant Virtual Switch	No single point of failure	Good
Virtual Switch Link aggregation[a]	No single point of failure	Very good (increase depends on number of OSA cards)

a. See 11.5, "z/VM VSWITCH link aggregation" on page 212, for details.

11.4 Network configuration parameters

In this section we review the most commonly used supported network drivers. This information will help you analyze which parameters to use to improve configuration and tuning.

11.4.1 Qeth device driver for OSA-Express

Queued Direct I/O (QDIO) is a highly efficient data transfer mechanism that satisfies the increasing volume of TCP/IP applications and increasing bandwidth demands. It reduces system overhead and improves throughput. These are QDIO components:

- Direct Memory Access
- Priority queuing
- Dynamic OSA address table update
- LPAR-to-LPAR communication
- IP assist functions

The qeth network device driver supports System z OSA-Express and Osa-Express2 features in QDIO mode and HiperSockets. This module is the most commonly used for network connection in Linux for System z in both physical and virtual networking configurations.

The qeth device driver requires three I/O subchannels for each CHPID in QDIO mode. One subchannel is for control reads, one for control writes, and the third for data. This device uses the QDIO protocol to communicate with the adapter (OSA or Hipersockets). Figure 11-7 shows the I/O subchannel interface.

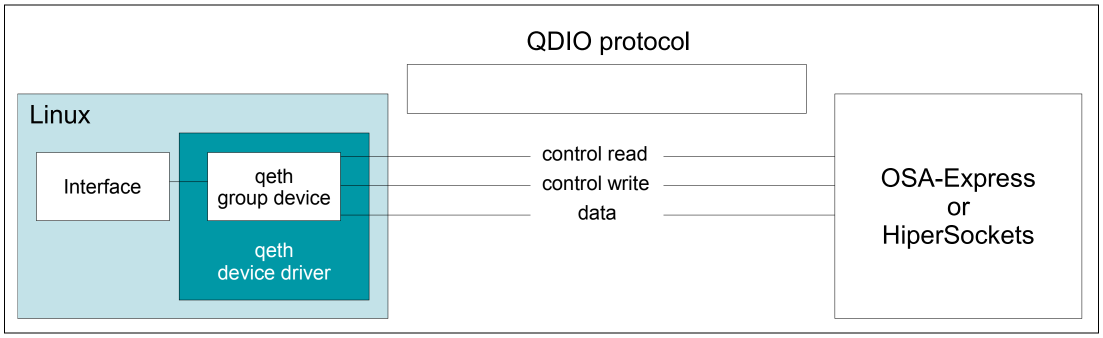


Figure 11-7 I/O subchannel interface

You can find details about the sysfs file system drivers from `/sys/bus/ccwgroup/drivers/qeth`.

You can modify, add, or query drivers parameters. Example 11-1 shows the file system structure related to the channel 0.0.0600 used in our test environment.

Example 11-1 List command to show sysfs files related to driver

```
#lnxwas:~ # ls -la /sys/bus/ccwgroup/drivers/qeth/0.0.0600/
total 0
drwxr-xr-x 8 root root    0 Sep 12 12:37 .
drwxr-xr-x 4 root root    0 Sep 12 12:37 ..
drwxr-xr-x 2 root root    0 Sep 13 11:30 blkt
-rw-r--r-- 1 root root 4096 Sep 12 12:37 broadcast_mode
-rw-r--r-- 1 root root 4096 Sep 12 12:37 buffer_count
-rw-r--r-- 1 root root 4096 Sep 12 12:37 canonical_macaddr
-r--r--r-- 1 root root 4096 Sep 12 12:37 card_type
lrwxrwxrwx 1 root root    0 Sep 14 16:09 cdev0 -> ../../css0/0.0.0000/0.0.0600
lrwxrwxrwx 1 root root    0 Sep 14 16:09 cdev1 -> ../../css0/0.0.0001/0.0.0601
lrwxrwxrwx 1 root root    0 Sep 14 16:09 cdev2 -> ../../css0/0.0.0002/0.0.0602
-rw-r--r-- 1 root root 4096 Sep 12 12:37 checksumming
-r--r--r-- 1 root root 4096 Sep 14 16:09 chpid
```

```
lrwxrwxrwx 1 root root    0 Sep 12 12:37 driver ->
../../../bus/ccwgroup/drivers/qeth
-rw-r--r-- 1 root root 4096 Sep 12 12:37 fake_broadcast
-r--r--r-- 1 root root 4096 Sep 14 16:09 if_name
-r--r--r-- 1 root root 4096 Sep 14 16:09 inbuf_size
drwxr-xr-x 2 root root    0 Sep 13 11:30 ipa_takeover
-rw-r--r-- 1 root root 4096 Sep 14 16:09 isolation
-rw-r--r-- 1 root root 4096 Sep 12 12:37 large_send
-rw-r--r-- 1 root root 4096 Sep 12 12:37 layer2
drwxr-xr-x 3 root root    0 Sep 12 12:37 net
-rw-r--r-- 1 root root 4096 Sep 12 12:37 online
-rw-r--r-- 1 root root 4096 Sep 14 16:09 performance_stats
-rw-r--r-- 1 root root 4096 Sep 12 12:37 portname
-rw-r--r-- 1 root root 4096 Sep 12 12:37 portno
drwxr-xr-x 2 root root    0 Sep 13 11:30 power
-rw-r--r-- 1 root root 4096 Sep 12 12:37 priority_queueing
--w------- 1 root root 4096 Sep 14 16:09 recover
-rw-r--r-- 1 root root 4096 Sep 12 12:37 route4
-rw-r--r-- 1 root root 4096 Sep 12 12:37 route6
drwxr-xr-x 2 root root    0 Sep 13 11:30 rxip
-rw-r--r-- 1 root root 4096 Sep 14 16:09 sniffer
-r--r--r-- 1 root root 4096 Sep 12 12:37 state
lrwxrwxrwx 1 root root    0 Sep 12 12:37 subsystem -> ../../../bus/ccwgroup
-rw-r--r-- 1 root root 4096 Sep 12 12:37 uevent
--w------- 1 root root 4096 Sep 14 16:09 ungroup
drwxr-xr-x 2 root root    0 Sep 13 11:30 vipa
lnxwas:~ #
```

Table 11-2 shows all parameters available on the sysfs file system, which can all be dynamically updated. The parameters are divided in the three different categories below, and described in Table 11-2:

- Display: To get information about the OSA configuration
- Configuration: To configure other features on OSA
- Performance: To tune the OSA adapter

Table 11-2 qeth device parameter

File	Category	Description
broadcast_mode	Configuration	For token ring broadcast.
buffer_count	Performance	You can assign from 8 to 128 buffers for inbound traffic.
canonical_macaddr	Configuration	For token ring networks.
card_type	Display	Display the card type used.
checksumming	Performance	Offload RX checksum calculation.
chpid	Display	The channel path ID used for this device.
fake_broadcast	Configuration	Add broadcast attribute to non-broadcast network.
if_name	Display	Linux interface name (for example, eth0).
inbuf_size	Configuration	Size of inbound buffer.

File	Category	Description
isolation	Configuration	Isolation policy of data connections.
large_send	Performance	Offload the TCP segmentation (starting with Osa-Express2 card).
layer2	Configuration	Enable layer2 including LLC header.
online	Configuration	Set online/offline device.
performance_stats	Configuration	Start and stop QETH performance statistics.
portname	Configuration	Display/modify OSA portname.
portno	Configuration	Specify the relative port number.
priority_queueing	Performance	Give a different priority queue in main storage for outgoing IP packets.
recover	Configuration	Recover a device in case of failure.
route4	Configuration	Set up a Linux router IPV4.
route6	Configuration	Set up a Linux router IPV6.
sniffer	Configuration	For Promiscuous mode.
state	Display	Display status of network.
ungroup	Configuration	To remove a qeth group device.

Use the **lsqeth** command to display your OSA configuration details. Example 11-2 shows output generated by a qdio driver used in a z/VM guest LAN scenario.

Example 11-2 lsqeth command output for guest LAN

```
# lsqeth
Device name                     : eth0
---------------------------------------------
        card_type               : GuestLAN QDIO
        cdev0                   : 0.0.0600
        cdev1                   : 0.0.0601
        cdev2                   : 0.0.0602
        chpid                   : 02
        online                  : 1
        portno                  : 0
        route4                  : no
        route6                  : no
        checksumming            : sw checksumming
        state                   : UP (LAN ONLINE)
        priority_queueing       : always queue 2
        fake_broadcast          : 0
        buffer_count            : 16
        layer2                  : 0
        large_send              : no
        isolation               : none
        sniffer                 : 0
```

Running QETH performance statistics

You can use the device group `/sys/bus/ccwgroup/drivers/qeth/<device_bus_id>/performance_stats` to collect performance data per device group. The default is 0. That is, no performance data gets collected. To start the collection of performance data, write 1 to the attribute (Example 11-3).

Example 11-3 Enable performance statistics collection

```
echo 1 > /sys/bus/ccwgroup/drivers/qeth/0.0.0600/performance_stats
```

Example 11-4 shows how to stop and reset the collection of all counters

Example 11-4 Disable and reset performance statistics collection

```
echo 0> /sys/bus/ccwgroup/drivers/qeth/0.0.0600/performance_stats
```

Use the **`ethtool`** command to obtain statistics (Example 11-5). For parameter details, refer to the **`ethtool`** tool menu.

Example 11-5 Performance statistics output sample using various parameters

```
# lnxwas:~ # ethtool -i eth0
driver: qeth_l3
version: 1.0
firmware-version: V611
bus-info: 0.0.0600/0.0.0601/0.0.0602

lnxwas:~ # ethtool -S eth0
NIC statistics:
     rx skbs: 10532457
     rx buffers: 9122180
     tx skbs: 11399530
     tx buffers: 11362492
     tx skbs no packing: 11356519
     tx buffers no packing: 11356519
     tx skbs packing: 43011
     tx buffers packing: 5973
     tx sg skbs: 0
     tx sg frags: 0
     rx sg skbs: 0
     rx sg frags: 0
     rx sg page allocs: 0
     tx large kbytes: 0
     tx large count: 0
     tx pk state ch n->p: 4383
     tx pk state ch p->n: 4383
     tx pk watermark low: 2
     tx pk watermark high: 5
     queue 0 buffer usage: 0
     queue 1 buffer usage: 0
     queue 2 buffer usage: 0
     queue 3 buffer usage: 0
     rx handler time: 16999574
     rx handler count: 8835917
     rx do_QDIO time: 243259
     rx do_QDIO count: 1140272
     tx handler time: 7844643
     tx handler count: 10221773
     tx time: 143818391
     tx count: 11399530
     tx do_QDIO time: 127463917
     tx do_QDIO count: 11362492
     tx csum: 0
     tx lin: 0
lnxwas:~ #
```

OSA parameter: buffer_count

The qeth device driver assigns 16 buffers by default for inbound traffic to each qeth group device. The defaults are not suitable in most cases, and defaults are already increased in the development stream patches provided on the DeveloperWorks website. Depending on the amount of available storage and traffic, you can assign from 8 to 128 buffers.

To understand Linux memory consumption, in case you are modifying this parameter, consider that Linux memory usage for inbound data buffers for the devices is:

```
(number of buffers) x (buffer size)
```

The buffer size is equivalent to the frame size, which is:

- For an OSA-Express CHPID in QDIO mode: 64 KB
- For Hipersockets: Depending on the Hipersockets CHPID definition, 16 KB, 24 KB, 40 KB, or 64 KB.

Set the buffer_count attribute to the number of inbound buffers that you want to assign. Example 11-6 shows the command used to modify the buffer_count parameter.

Example 11-6 Command used to modify the buffer_count parameter

```
echo 64 > /sys/bus/ccwgroup/drivers/qeth/0.0.0600/buffer_count
```

Note: The device must be offline while you specify the number of inbound buffers, otherwise the write operation is not permitted.

OSA parameter: checksumming

You can omit checksumming. Otherwise, perform it via software (for example, TCP/IP stack) or hardware, if the CHPID is an OSA-Express CHPID in QDIO mode and your OSA adapter hardware supports checksumming. To check how checksumming is performed, issue a command where device 0.0.0600 is assumed to be used (Example 11-7).

Example 11-7 Display checksumming on device

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.0600/checksumming
sw checksumming
```

To change the checksumming parameter, use the **echo** command to add the checksumming file value. These are possible values:

- `hw_checksumming`: Checksumming is performed by the OSA adapter. If `hw_checksumming` is specified but not supported, the TCP/IP stack performs checksumming instead.
- `sw_checksumming`: Default. Checksumming is performed by the TCP/IP stack.
- `no_checksumming`: Checksumming is suppressed. This might jeopardize data integrity.

This functionality will move to the **ethtool** command in future development with newer kernels.

Note: The device must be offline while you change the checksumming parameter.

OSA parameter: large_send

You can offload the TCP segmentation from the Linux network stack to OSA-Express2 and OSA-Express3 in layer 3 mode. A large send can lead to increased performance and latency for an interface with a high amount of large outgoing packets. Use the **echo** command to modify the `large_send` parameter (Example 11-8).

Example 11-8 echo command to modify the large_send parameter

```
echo <value> > /sys/bus/ccwgroup/drivers/qeth/0.0.0600/large_send
```

Where <*value*> can be one of these:

- no: No large send is provided. This is the default. The Linux TCP/IP stack performs the segmentation.
- TSO: The OSA network adapter performs segmentation.
- EDDP: The qeth driver performs segmentation.

There is already support of large receive provided in the development stream and in the RHEL6.1 distribution.

OSA parameter: priority_queuing

An OSA-Express CHPID in QDIO mode has four output queues in main storage, giving these queues different priorities. Queuing is relevant to high-traffic situations. Use the **cat** command to check the priority_queuing. Example 11-9 shows the priority queuing set in our 0.0.c200 device.

Example 11-9 Display priority queue related to 0.0.c200 device

```
# cat /sys/bus/ccwgroup/drivers/qeth/0.0.c200/priority_queueing
always queue 2
```

You can modify the OSA parameter using these values:

- `prio_queuing_prec`: To base the queue assignment on the two most significant bits of each packet's IP header precedent field.
- `prio_queuing_tos`: To select a queue according to the IP type of service assigned by an application to packets (auto managed queue).
- `no_prio_queueing`: Queue 2 is used for all packets. This is the default.
- `no_prio_queueing:0`: Queue 0 is used for all packets.
- `no_prio_queueing:1`: Queue 1 is used for all packets.

- `no_prio_queueing:2`: Queue 2 is used for all packets.
- `no_prio_queueing:3`: Queue 3 is used for all packets.

Table 11-3 shows the relation between the queue and the service type, and details how the qeth device driver maps service types to the availability queues in the `prio_queueing_tos` parameter.

Table 11-3 IP service types and queue assignment for type of service queuing

Service type	Queue
Low latency	0
High throughput	1
High reliability	2
Not important	3

Note: The device must be offline while you change the priority queue parameter. This function applies to SA-Express CHPIDs in QDIO mode only.

11.4.2 LAN channel station (non-QDIO)

The LAN channel station device driver (LCS device driver) supports OSA features in non-QDIO mode. If you use LCS drivers, you can share the adapter between images, but you need to configure it via OSASF. With this device driver, the OAT table is not dynamically managed, so you need to complete the appropriate configurations. Figure 11-8 shows the I/O subchannel interface.

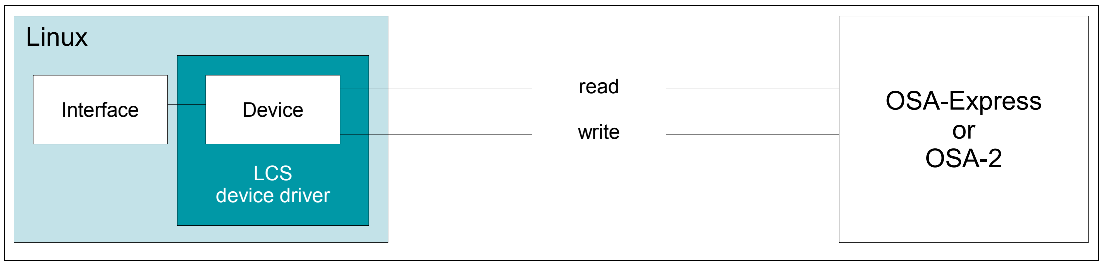


Figure 11-8 I/O subchannel interface

11.4.3 CTCM device driver

The CTCM driver provides ESCON® and FICON Channel to Channel (CTC) connections and virtual CTCA connections between guests hosted on the same z/VM system. It also provides CTC-based Multi-Path Channel (MPC) connections that are needed by SLES11 SP1 to communicate with VTAM® on traditional mainframe operating systems. Use the CTC connections only used for migration as they are depreciated. Figure 11-9 shows the I/O subchannel interface.

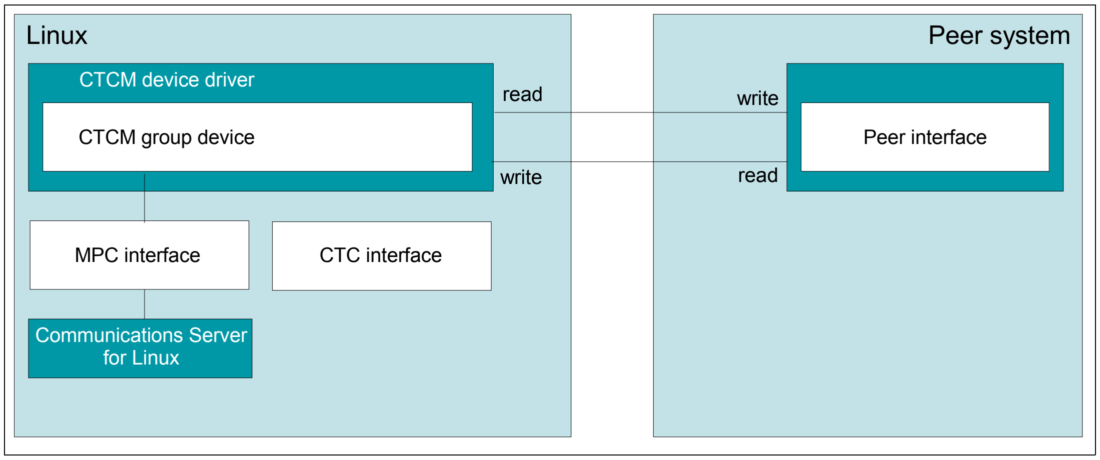


Figure 11-9 I/O subchannel interface

11.4.4 Other OSA drivers

Additional drivers are also available, for example, NETIUCV and Common Link Access to Workstation (CLAW). Because these drivers are depreciated, use them only for migration.

11.5 z/VM VSWITCH link aggregation

Link aggregation support for the virtual switch was first introduced in z/VM Release 5.3. The virtual switch can be configured to operate with aggregated links in concert with a switch that also supports the IEEE 802.3ad link aggregation specification. Aggregating links with a partner switch box allows multiple OSA-Express2 or OSA-Express3 adapters to be deployed to transfer data between the switches. This configuration provides the benefits of increased bandwidth and near seamless recovery of a failed link within the aggregated group. Figure 11-10 shows the architecture based on z/VM Virtual Switch Link Aggregation.

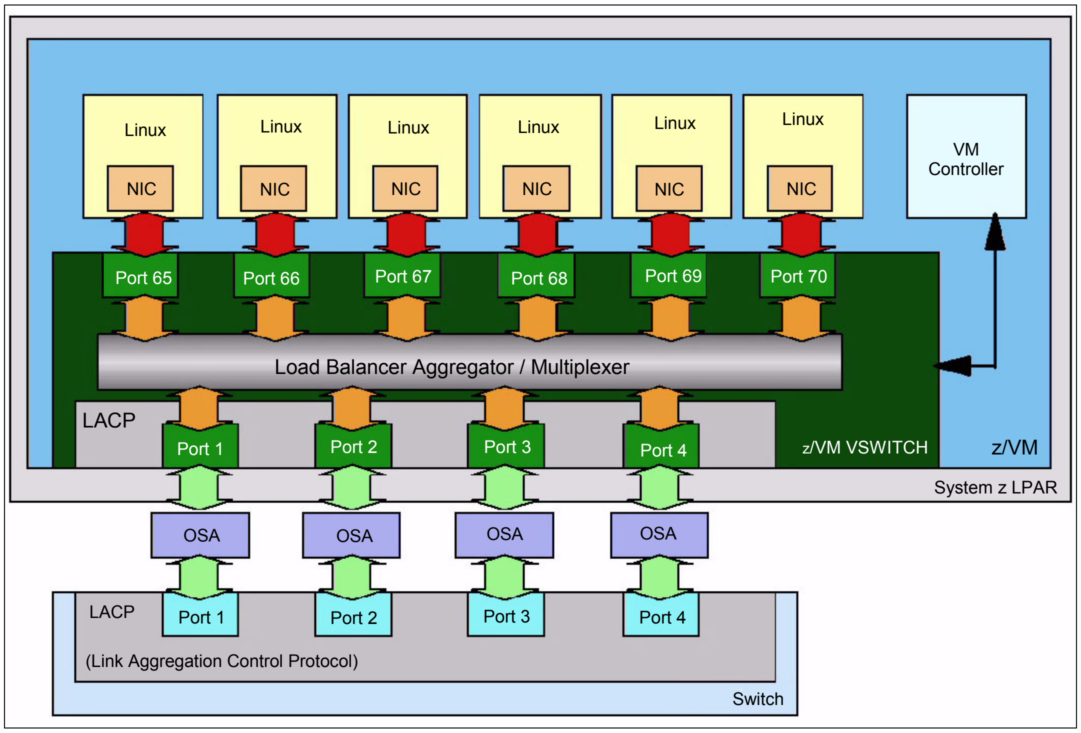


Figure 11-10 z/VM VSWITCH link aggregation architecture

11.6 HiperSockets

HiperSockets is a technology that provides high-speed TCP/IP connectivity between servers within a System z. This technology eliminates the requirement for any physical cabling or external networking connection among these servers. HiperSockets works as an internal LAN, secure and protected, with connections that work synchronously. HiperSockets' largest advantage is the enormous provided bandwidth and the high security. HiperSockets consume processor cycles, because they run as emulated networks. However, they can improve application performance, especially in solutions based on three-tiered architecture. Figure 11-11 shows an example of a three-tier architecture.

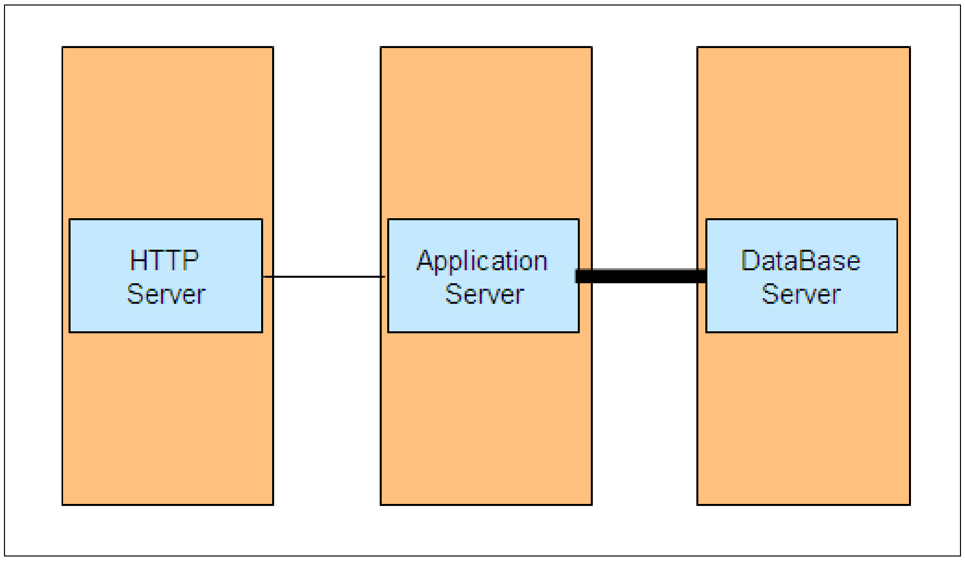


Figure 11-11 Three-tier architecture

Cascades of web servers, SAP application servers, and database servers are examples of a three-tier architecture. This architecture might face network performance problems when you need a high-backend throughput. In fact, many applications might require a dedicated network for performance reasons between business logic and data. HiperSockets provides this connectivity using an internal queued input/output (iQDIO) at memory speed to pass traffic among servers in the mainframe. Another HiperSocket advantage, in addition to the high maximum throughput, is security. You do not need firewalls or encryption within one physical mainframe. You also do not need checksumming because the memory integrity is guaranteed by the hardware. Also, no physical switches, which are expensive, or cables, which are error prone, are needed.

Figure 11-12 displays the difference between OSA cards and HiperSockets in a STRG benchmark test. Notice the advantages of the HiperSockets connection over the OSA-cards in terms of throughput and costs per transferred MB of data.

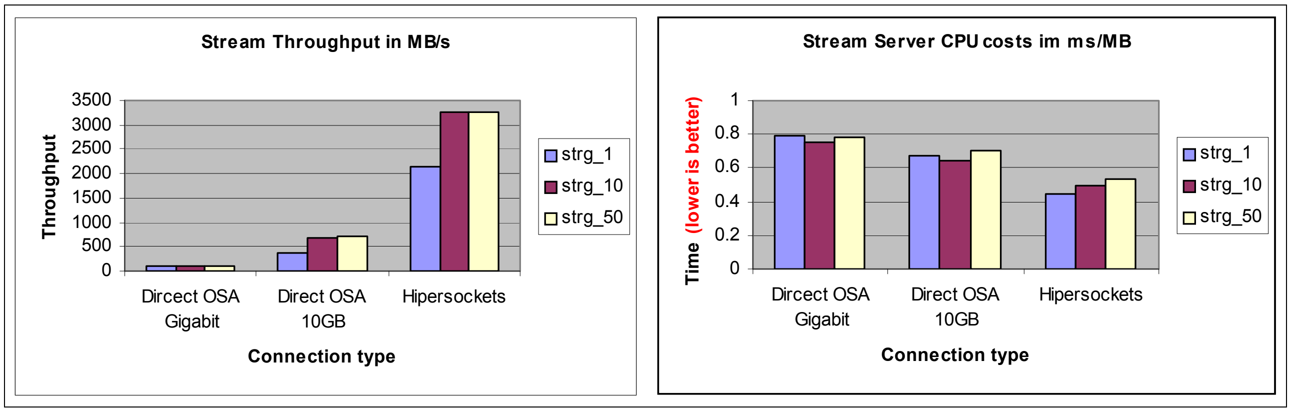


Figure 11-12 Direct Attached OSA 1 GBit and 10 GBit versus HiperSockets

To learn more, view a presentation about HiperSockets checksumming and connections to z/OS and SAP servers at this link:

`http://public.dhe.ibm.com/software/dw/linux390/perf/network_tuning.pdf`

11.7 High-availability costs

We have noted that network availability is one of the most common problems related to servers. When you plan a Linux architecture on System z, consider potential single points of failure. Networking high-availability scenarios might influence the overall performance of systems. In z/VM, you can use VSWITCH or Link aggregation support for high-availability, depending on your System z machine, z/VM level, and network topology. With Linux on LPAR, you need to manage network failure with Linux software (for example, zebra), which provides daemons to manage virtual IP addressing and dynamic routing. The following sections discuss the costs of high-availability networking on Linux systems in LPARs or as a z/VM guest.

LPAR: Static routing versus dynamic routing

In LPAR scenarios, you can use Virtual IP Address (VIPA) to architect a high-availability solution for adapters. VIPA allows you to assign IP addresses that are not associated with a particular adapter. The address is routed to one real interface. You can create static routing or use dynamic routing to communicate with protocols in the network and update the routing table.

Many distributions support quagga as the dynamic routing manager. Quagga use two daemons:

- zebra

  This provides kernel routing updates and redistributes routes between various routing protocols and interface lookups.

- ospfd

  Open shortest path first is a routing protocol classified as an interior gateway protocol. It distributes routing information between routers belonging to a single autonomous system.

Using a daemon for dynamic routing introduces the workload to be managed. Daemons like quagga wake up periodically, sending packets to neighbors to maintain connectivity. You set up the wake-up interval on the hello-interval parameter, but remember that it can influence overall performance.

Dynamic routing is important if you need auto-managed high availability. If you use a static route, you must manually modify the routing configuration whenever a signal loss or other OSA interface card related problems occur. Note that the time (from when the problem is reported to the time that you manually modify the static route) can affect application availability and user point of view.

In our next test, we explore the differences between static and dynamic routes in terms of throughput and CPU usage, using connect request response reports. The test was conducted in 2007, so the results do not reflect current capabilities but are still valuable in analyzing a trend. Figure 11-13 shows the differences in transactions per second. Static routing performs better than the other two cases. There are no significant differences in dynamic routing provided that you change the hello-interval parameter, indicated by the number in parentheses in the graph legend.

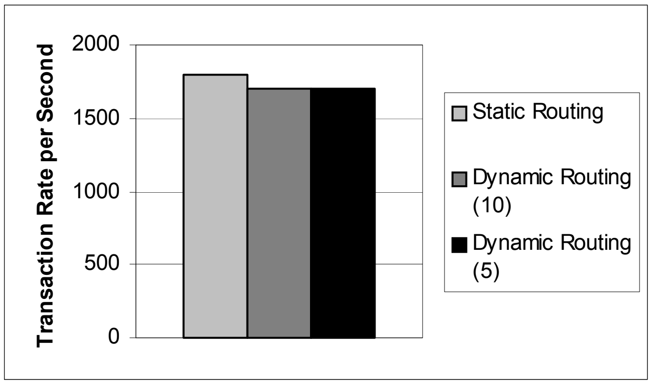


Figure 11-13 Transaction rate per second

Notice the differences in CPU usage. Continuos polling hello-interval affects the CPU used to manage connections. Figure 11-14 shows these differences.

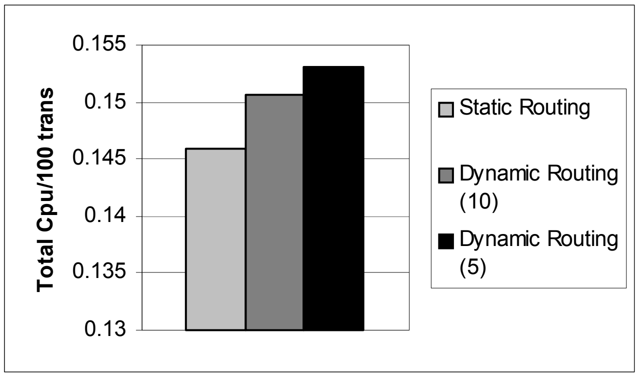


Figure 11-14 Total CPU/100 trans

11.8 Performance tools

We used various workload scenarios for the experiments and tests in this chapter. In this scenario, we reproduced real cases of production network traffic, using these tools to documents the results:

- netperf

  This tool is intended to measure performance of bulk, unidirectional streaming, and to request response network traffic using a client server model. Netserver runs on the server, and netperf runs on the client, controlling the benchmark. For more details, see:

  `http://netperf.org/netperf/`

- iperf

  This tool is based on a client server model and measures maximum UDP and TCP bandwidth. For more information, see:

  `http://iperf.sourceforge.net/`

- tracepath [destination]

  This command checks the MTU size to final the destination.

  Note: Set the MTU size to the maximum size supported by all hops on the path to the final destination to avoid fragmentation.

We used these workloads in our tests:

- Request-response test (RR)

  RR describes data exchange that usually occurs in interactive environments, that is, long-running transport connections with small amounts of data being exchanged in a conversational way. Here, *networking performance* indicates transactions per wall clock second and CPU time consumed per transaction.

- Connect-request-response (CRR)

  CRR describes data exchange that usually occurs with a HTTP server, namely, that at connection start, a small amount of data flows in, a moderate amount flows out, and the connection then ends. For this workload, we are still interested in transactions per wall clock second and CPU time consumed per transactions.

- Streaming get (STRG)

  In STRG, the connection is persistent, just as in RR. However, the data sent in one direction is very small (that is, a few bytes), and the data sent in the other direction is very large. This scenario illustrates the best-case rate at which the computing system can pump data across a communication link. We measured processor costs and the throughput in megabytes per second. Here we examined megabytes per wall clock second and CPU time used per megabyte data sent.

11.9 General network performance information

You need to adapt many parameters to network connection needs depending on the workload characteristics, the underlying hardware features, connected operating systems, (for example, Linux to Linux or Linux to z/OS), and Linux versions. Remember that the default values change with different releases.

Use this checklist to help optimize your network performance:

- Choose the correct connection type. View a comprehensive presentation here:

  http://public.dhe.ibm.com/software/dw/linux390/perf/network_connection_types.pdf

- Choose the correct parameters.
  - Choose the MTU size carefully. Use MTU 8992, if supported, to increase throughput and to save cycles.
  - Adapt inbound and outbound window size to suit the workload.
  - Inbound buffer count can be increased to 128.
  - Consider switching on priority queueing with the OSA card.
- Switch off unneeded daemons.
- Switch off firewalls in controlled environments without external connections, such as HiperSockets.
- Switch off checksumming for HiperSocket connections, as they are protected by the ECC mechanism.

For more details and tips about optimizing network connections, see this website:

http://www.ibm.com/developerworks/linux/linux390/perf/tuning_networking.html

A

WebSphere performance benchmark sample workload

In this publication, we simulated measurable workloads for each example using the Trade Performance Benchmark Sample for WebSphere Application Server Version 6.1.0.11 (Trade 6) as the sample application. This appendix describes the sample application.

IBM Trade Performance Benchmark sample

The IBM Trade Performance Benchmark sample, also known as the Trade 6 application, is a sample WebSphere end-to-end benchmark and performance sample application. This benchmark, designed and developed to cover the WebSphere programming model, provides a real-world workload, driving WebSphere's implementation of J2EE 1.4 and web services, including key WebSphere performance components and features.

Trade 6 simulates a stock trading application that allows you to buy and sell stock, check your portfolio, register as a new user, and so on. We used Trade 6 to generate workloads that we can analyze in terms of their impact on system performance. You can download the IBM Trade Performance Benchmark sample for WebSphere Application Server, for free, here:

`https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=trade6`

These are the main Trade 6 components are listed below (Figure A-1):

- IBM HTTP server

  The HTTP server accepts client requests and delivers static content, for example, HTML pages, images, and style sheets. Dynamic requests are forwarded to the WebSphere Application Server through a server plug-in.

- IBM WebSphere Application Server

  This server creates dynamic content using JavaServer Pages (JSPs) and Java Servlets. Pages are generated from data extracted from a DB2 database. All Trade versions are WebSphere-version dependent, so Trade 6 only works with WebSphere Application Server V6.

- DB2 database

  The DB2 database contains relational tables of simulated customer accounts and stock transactions. We used DB2 LUW v8.2.

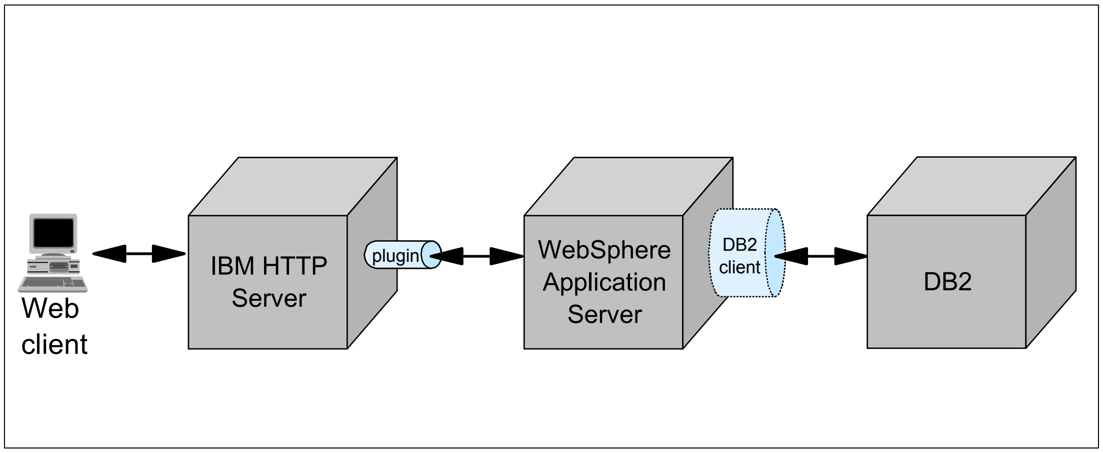


Figure A-1 Components of a typical Trade 6 deployment

Trade 6 Deployment options

You can use any of these options to deploy Trade 6 on Linux for System z:

- All components can run in a single Linux guest, which is referred to as a single-tier deployment.
- Each component can run in a dedicated Linux guest, referred to as three-tier deployment.

We chose a three-tier deployment for this book, running the IBM HTTP server, the WebSphere Application Server, and DB2 in their own Linux guests (Figure A-2).

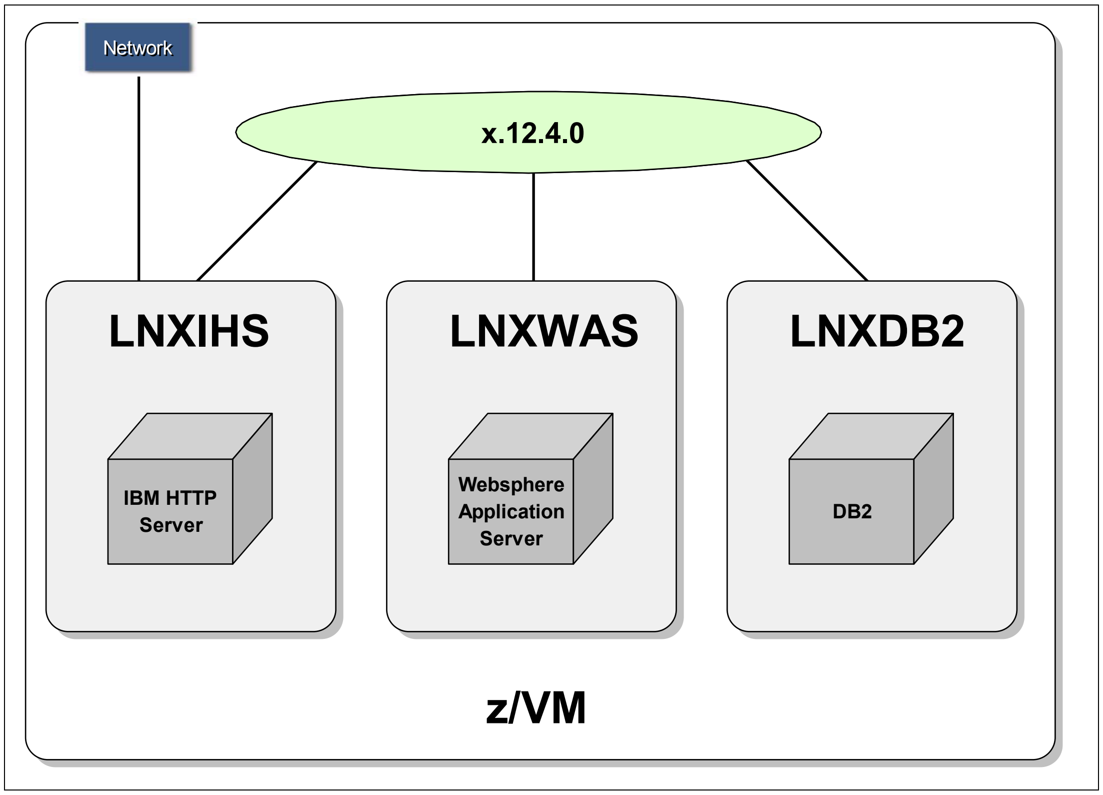


Figure A-2 Three-tier deployment

The three-tier deployment allowed us to more accurately adjust the virtual machine size of each Linux guest, based on its task. The deployment also allowed us to attribute specific resource usage to a specific task.

B

WebSphere Studio Workload Simulator

This appendix describes our use of the WebSphere Studio Workload Simulator as a workload generator. Topics discussed include an overview of the WebSphere Studio Workload Simulator and the sample workload generation script that we used in conjunction with the Trade 6 WebSphere application. See Appendix A, “WebSphere performance benchmark sample workload” on page 219, for more information about Trade 6.

WebSphere Studio Workload Simulator overview

The WebSphere Studio Workload Simulator for z/OS and OS/390 allows you to create multiple `virtual` or `simulated` users to test load and performance.

The Workload Simulator consists of two components:

- A controller
- An engine

For high scalability, the engine, which generates the load used during load-testing, is installed on a System z server. The load generated by the engine can be used to test any web-serving environment. That is, the environment to be tested is not limited to z/OS. In our case, we tested against a WebSphere Application Server that was running on a Linux on System z guest. Workload Simulator supports multiple engines.

Sample workload generation script

This section outlines the script that we used to generate a workload against the WebSphere Application Server. See Appendix C, “Additional material” on page 233, for information about how to download the script that we used. You can also download the `MultiClients.conf` configuration file, and you also need to create an engine definition. After you download the script (Example B-1), modify the string host name to reference your own host name.

Example: B-1 Sample Trade 6 workload generation script

```
//
/*trade6 version 1.2*/
init_section
{
   string hostname = "x.12.4.xxx:9080";
   //string hostname = "x.12.4.xxx";
   int botClient = 0;
   int topClient = 3999;
   shared int curClient = botClient - 1;
}
int html_percent = 10;
int gif_percent = 0;
//repopulate and reconfigure trade on the website to reflect the following numbers
int num_u = 4000;
int num_s = 2000;
int num_stock, i;
string name, uid, add, email, holdid, stocks;
bool loop = true;
int sell_deficit = 0;
HttpResponse r;
start_transaction("login");
   startpage(4);
   thinktime(1000);
      //uid = URLEncode("uid:"+random(0,num_u - 1));
      //TODO: Make this random but better than above - must be random across all
clients
      int clientnum;
      enter_critical_section();
      curClient = curClient + 1;
      if (curClient > topClient)
      {
         curClient = botClient;
      }

      clientnum = curClient;
      leave_critical_section();
      uid = URLEncode("uid:" + clientnum);
      postpage(""+hostname+"","/trade/app",1,close,0,start,"",
      "uid=" +uid+ "&passwd=xxx&action=login",
      "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
      "Referer: http://"+hostname+"/trade/app",
      "Accept-Language: en-us",
      "Content-Type: application/x-www-form-urlencoded",
      "Accept-Encoding: gzip, deflate",
      "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
      "Host: "+hostname+"",
      "Connection: Keep-Alive");

   endpage;
end_transaction("login");
while (loop)
{
   distribute
   {
      weight 20:
         start_transaction("home");
            startpage(5);
            thinktime(1000);
               getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=home",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
         end_transaction("home");

      weight 4:
         start_transaction("update_account");
            // Performan an account, followed by an account update
            startpage(6);
            thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=account",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Content-Type: application/x-www-form-urlencoded",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
                "Host: "+hostname+"",
                "Connection: Keep-Alive");

             endpage;
             // update the account to some random data
             startpage(7);
             thinktime(5000);
                getpage(""+hostname+"","/trade/app",1,close,0,start,"?userID="
+uid+ "&fullname=" +"rnd"+current_client()+now()+
"&password=xxx&address=rndAddress&cpassword=xxx&creditcard=rndCC&email=rndEmail&ac
tion=update_profile",
                "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
                "Referer: http://"+hostname+"/trade/app",
                "Accept-Language: en-us",
                "Accept-Encoding: gzip, deflate",
                "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
                "Host: "+hostname+"",
                "Connection: Keep-Alive");

             endpage;
          end_transaction("update_account");

       weight 10:
          start_transaction("account");
             startpage(6);
             thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=account",
                "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
                "Referer: http://"+hostname+"/trade/app",
                "Accept-Language: en-us",
                "Accept-Encoding: gzip, deflate",
                "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
                "Host: "+hostname+"",
                "Connection: Keep-Alive");

             endpage;
          end_transaction("account");

       weight 4:
          start_transaction("sell");
             startpage(8);
             thinktime(1000);
                r =
getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=portfolio",
                "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
                "Referer: http://"+hostname+"/trade/app",
                "Accept-Language: en-us",
                "Accept-Encoding: gzip, deflate",
                "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
                "Host: "+hostname+"",
                "Connection: Keep-Alive");

             endpage;
             holdid = r.extractVariable("holdingID=","\"");
```

```
            if (StrLength(holdid) > 0)
            {
               startpage(9);
               thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=sell&holdingID="
+holdid,
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

               endpage;
            }
            else
            {
               // If the user has no holdings, switch to a buy and increment
sell_deficit counter
               sell_deficit = sell_deficit + 1;
               stocks="s:"+random(0, num_s-1);
               startpage(10);
               thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=quotes&symbols="
+stocks,
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

               endpage;
               startpage(11);
               thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=buy&symbol=" +stocks+
"&quantity=" +random(1,
                  200),
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

                endpage;

              }

          end_transaction("sell");

        weight 12:
          start_transaction("portfolio");
            startpage(8);
            thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=portfolio",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
          end_transaction("portfolio");

        weight 4:
          stocks="s:"+random(0, num_s-1);
          start_transaction("buy");
            if ( sell_deficit <= 0 )
            {
              startpage(10);
              thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=quotes&symbols="
+stocks,
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

              endpage;
              startpage(11);
              thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=buy&symbol=" +stocks+
"&quantity=" +random(1,
                  200),
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                  "Referer: http://"+hostname+"/trade/app",
                "Accept-Language: en-us",
                "Accept-Encoding: gzip, deflate",
                "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                "Host: "+hostname+"",
                "Connection: Keep-Alive");

              endpage;
            }
            else
            {
              // If there is a sell deficit, perform a sell instead to keep
buys/sells even
              startpage(8);
              thinktime(1000);
                r =
getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=portfolio",
                "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                "Referer: http://"+hostname+"/trade/app",
                "Accept-Language: en-us",
                "Accept-Encoding: gzip, deflate",
                "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                "Host: "+hostname+"",
                "Connection: Keep-Alive");

              endpage;
              holdid = r.extractVariable("holdingID=","\"");
              if (StrLength(holdid) > 0)
              {
                sell_deficit = sell_deficit - 1;
                startpage(9);
                thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=sell&holdingID="
+holdid,
                  "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
*/*",
                  "Referer: http://"+hostname+"/trade/app",
                  "Accept-Language: en-us",
                  "Accept-Encoding: gzip, deflate",
                  "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.1)",
                  "Host: "+hostname+"",
                  "Connection: Keep-Alive");

                endpage;
              }


            }

          end_transaction("buy");
```

```
      weight 40:
         start_transaction("quotes");
            stocks="s:"+random(0, num_s-1);
            startpage(10);
            thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=quotes&symbols="
+stocks,
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
         end_transaction("quotes");

      weight 2:
         start_transaction("register");
            //log the current user out and register a new user
            startpage(12);
            thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=logout",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
            clear_cookie_cache();
            // load the registration page
            startpage(13);
            thinktime(1000);
               getpage(""+hostname+"","/trade/register.jsp",1,close,0,start,"",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
            // register a new user and login
            name = URLEncode("first:"+random(0,999)+" last:"+random(0,4999));
            add = URLEncode(random(1, 5000) + " mystreet");
            uid = URLEncode("ru:"+current_client()+":"+now());
            email = URLEncode(uid+"@"+random(0,100)+".com");
            startpage(14);
            thinktime(1000);
               getpage(""+hostname+"","/trade/app",1,close,0,start,"?Full+Name="
+name+ "&snail+mail=" +add+ "&email=" +email+ "&user+id=" +uid+
"&passwd=yyy&confirm+passwd=yyy&money=1000000&Credit+Card+Number=123-fake-ccnum-45
6&action=register",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
         end_transaction("register");

      weight 4:
         start_transaction("logoff");
            startpage(12);
            thinktime(1000);

getpage(""+hostname+"","/trade/app",1,close,0,start,"?action=logout",
               "Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*",
               "Referer: http://"+hostname+"/trade/app",
               "Accept-Language: en-us",
               "Accept-Encoding: gzip, deflate",
               "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)",
               "Host: "+hostname+"",
               "Connection: Keep-Alive");

            endpage;
            clear_cookie_cache();
         end_transaction("logoff");
         loop = false;

   }
}

stop;
```

Learn more about this product here:

http://www-306.ibm.com/software/awdtools/studioworkloadsimulator/about/

Contact your IBM representative for product cost and features.

C

Additional material

This book refers to additional material that can be downloaded from the internet as described below.

Locating the web material

The web material associated with this book is available in softcopy on the internet from the IBM Redbooks web server. Point your web browser at:

ftp://www.redbooks.ibm.com/redbooks/SG246926

Alternatively, you can go to the IBM Redbooks website at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG24-6926.

Using the Web material

The additional Web material that accompanies this book includes the following files:

File name	*Description*
SG24692602.zip	Zipped samples

System requirements for downloading the web material

The following system configuration is recommended to run the code. You might need more to deploy and run the Trade 6 application:

Hard disk space	3 MB minimum
Operating system	Windows or Linux

How to use the web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the web material zip file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 236. Note that some of the documents referenced here may be available in softcopy only.

- *Linux for IBM System z9 and IBM zSeries*, SG24-6694
- *Linux Performance and Tuning Guidelines*, REDP-4285
- *z/VM and Linux on IBM System z: The Virtualization Cookbook for SLES9*, SG24-6695
- *Linux on IBM eServer zSeries and S/390: Performance Toolkit for VM*, SG24-6059

Other publications

These publications are also relevant as further information sources:

- *z/VM Performance Toolkit Guide,* SC24-6156
- *z/VM Performance,* SC24-6109
- *z/VM Getting Started with Linux on System z,* SC24-6096
- *IBM Tivoli OMEGAMON XE on z/VM and Linux, User’s Guide,* SC32-9489
- *IBM Linux on System z - Device Drivers, Features, and Commands on SUSE Linux Enter*prise Server 11 SP1, SC34-2595-01
- *IBM Linux on System z - Device Drivers, Features, and Commands Development Stream,* SC33-8411-11

Online resources

These websites are also relevant as further information sources:

- Introduction to the New Mainframe: z/VM Basics

  `http://www.redbooks.ibm.com/redpieces/abstracts/sg247316.html?Open`
- IBM: VM performance documents

  `http://http://www.vm.ibm.com/perf/docs/`
- IBM: VM performance tips

  `http://www.vm.ibm.com/perf/tips/`

- IBM: Linux on System z - performance hints and tips

  `http://www.ibm.com/developerworks/linux/linux390/perf/index.html`
- IBM Linux on System z - development home page

  `http://www.ibm.com/developerworks/linux/linux390/index.html`

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

Linux on IBM System z: Performance Measurement and Tuning

Linux on IBM System z Performance Measurement and Tuning

Understanding Linux performance on System z

z/VM performance concepts

Tuning z/VM Linux guests

This IBM® Redbooks® publication discusses performance measurement and tuning for Linux® for System z™. It is intended to help system administrators responsible for deploying Linux for System z understand the factors that influence system performance when running Linux as a z/VM® guest.

This book starts by reviewing of the basics involved in a well-running Linux for System z system. This book also provides an overview of the monitoring tools that are available and that were used throughout this book.

Additionally, performance measurement and tuning at both the z/VM and the Linux level is considered. This book also offers tuning recommendations. Measurements are provided to help illustrate what effect tuning controls have on overall system performance.

The system used in the writing of this book is IBM System z10™ running z/VM Version 6.1 RSU 1003 in an LPAR. The Linux distribution used is SUSE Linux Enterprise Server 11 SP1. The examples in this book use the Linux kernel as shipped by the distributor.

The z10 is configured for:

- Main storage: 6 GB
- Expanded storage: 2 GB
- Minidisk cache (MDC): 250 MB
- Total LPARs: 45
- Processors: Four shared central processors (CPs) defined as an uncapped logical partition (LPAR)

The direct access storage devices (DASD) used in producing this publication are 2105 Enterprise Storage Server® (Shark) storage units.

The intent of this book is to provide guidance on measuring and optimizing performance using an existing zSeries® configuration. The examples 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 benchmarks.

SG24-6926-02 ISBN 0738436321