March 2011

Linux on IBM System z
Large Discontiguous Saved Segments (DCSS) Under Linux

Linux end to end Performance Team:
Dr. Juergen Doelle, David C. Sadler

Table of Contents

About this publication

This document provides results for tests run using large Discontiguous Saved Segments under Linux®. A DCSS is a z/VM technology used to share memory between a number of guests. This paper focuses on three areas of application for a large DCSS: sharing code, sharing read-only data, and using a DCSS as a swap device.

Acknowledgments

Special Thanks Xenia Tkatschow and Bill Bitner for their support and guidance with DCSS and z/VM related questions.

Introduction

When analyzing z/VM® system performance as it relates to the use of DCSS, it is important to understand some basic terms and concepts.

When Linux runs under z/VM, each Linux instance loads its own copies of programs, and manages its own storage. When multiple copies of Linux are running on the same z/VM system, there is an opportunity to use functions unique to z/VM and IBM System z® to share memory pages between the multiple Linux guests.

A *saved segment* is a special feature of z/VM that provides a range of virtual storage pages, which are defined to hold data or reentrant code (programs). The administrator can save code or data in saved segments, assign them a name, and dynamically attach or detach them from multiple virtual machines. There are two types of segments, whose names are derived from the command operands used to define them:

SN
Implements shared read/write access, no data saved.

SR
Implements shared read-only access.

A *Discontiguous Saved Segment (DCSS)* is a z/VM technology used to save a portion of the storage of a guest. This storage can then be shared between a number of guests, which all see the same storage. DCSS is a powerful tool that gives z/VM administrators a way to provide high levels of resource sharing between guests, combined with excellent performance. A DCSS occupies one or more memory segments (1 MB portions of real storage).

A DCSS can be created covering a contiguous range of addresses, or a discontiguous range, that is a number of different sections of the guest's address range. A DCSS can even have parts that are read/write, where each guest has its own copy of a portion of the shared

segment that the guest can modify. Linux can use DCSS technology to build file systems that reside in memory, drastically reducing file system I/O time for system files.

The Linux *eXecute-In-Place (XIP)* technology is used to treat code in a memory-backed file system as if it were a part of the virtual memory space. Instead of being loaded into Linux guest memory, executables residing in a DCSS can be executed directly from the DCSS memory. This reduces the amount of central storage required to host a group of Linux systems running similar workloads. Therefore, larger numbers of virtual Linux systems can run in a given system footprint.

A DCSS can consist of shared or exclusive segments:

- A *shared segment* is memory accessed by multiple virtual machines. Any change to that memory made by one virtual machine is instantaneously seen by all virtual machines accessing the segment.
- An *exclusive segment* is memory where each virtual machine has its own exclusive copy. In this case, changes made by one virtual machine are not seen by other virtual machines accessing the segment.

Whether a particular portion of a DCSS is read/write does not determine whether it is exclusive or shared.

Linux makes use of a DCSS by presenting it as a block device, in the same way that DASDs are presented for use. The *dcssblk device driver* is part of the Linux kernel and makes a DCSS appear as a block device. Even though a DCSS looks just like other block devices, dcssblk enables XIP supporting file systems to recognize that the DCSS is memory-addressable.

Current versions of the Linux *second-extended (ext2) file system* support XIP using a `mount` option. A DCSS-backed ext2 file system can provide this capability.

With z/VM 5.4, IBM announced the *large DCSS*. A DCSS can now include pages up to 512 GB in size, which exceeds the previous limitation of 2 GB. A large DCSS can reside almost anywhere in addressable storage. The Linux dcssblk device driver concatenates DCSSs of 2 GB, so that multiple DCSSs appear to Linux as a single device. This feature enables Linux to build very large DCSS-backed file systems, and eliminates the maintenance complexity of some of the earlier DCSS configuration approaches.

Linux kernel level 2.6.26 (shipped in SUSE Linux Enterprise Server (SLES) 11 from Novell) contains the updated dcssblk driver needed to handle large DCSSs. We recommend in any case the use of SLES11 SP1 and, due to some testing problems, the maintenance kernel update to 2.6.32.19 or higher, which is also for Red Hat Enterprise Linux 6.

The remainder of this paper refers only to large DCSSs, so the term 'DCSS' is used to mean a large DCSS.

The z/VM DEFSEG and SAVESEG commands are used to map pages of memory contents, and to store them in disk-backed spool space that can be made accessible to multiple virtual machines. These dcssblks provide disk-like access to the saved segments.

These features enable multiple Linux guests to share one copy of commonly run code, and reduce overall memory use by Linux guests.

In a virtual server farm with similar Linux instances, there is often a considerable amount of data that is required by all instances. Use of a DCSS enables z/VM to load such data into a designated segment of physical memory, and enables all Linux instances that need the data to share this memory segment.

For more information on DCSS, see
z/VM Saved Segments Planning and Administration

For particular information on the types of saved segments, see

- *Glossary*
- z/VM Saved Segments Planning and Administration, Types of Saved Segments

For more information on DCSS, dcssblk and its use, see
Managing Linux on System z: Using device drivers, features, and commands

Objectives

This paper focuses on three areas of application for a large DCSS: sharing code, sharing read-only data, and using a DCSS as a swap device.

Defining frequently used data and code such as licensed programs, as saved segments has several advantages:

- Because several users can access the same physical storage, real storage use is minimized.
- Using saved segments decreases the I/O rate and DASD paging space requirements, thereby improving virtual machine performance.
- Saved segments attached to a virtual machine can reside beyond the range of the virtual machine's defined virtual storage. Therefore, the virtual machine can use its defined storage for other purposes.

- Reentrant programs residing within the page ranges of a saved segment can be shared by concurrently operating virtual machines. Rarely used code can be placed in a saved segment and loaded into a virtual machine when needed.

This paper focuses on three areas of application for a large DCSS:

Sharing code

- Sharing code saves memory and shortens startup times, because the binaries are executed directly from the DCSS using the execute-in-place technology. The test results presented in this paper quantify the savings.
- The previous limitation of a DCSS to a maximum of 2 GB in size required much effort to determine the libraries suitable for sharing, and to maintain the consistency of the installation during updates.

The large DCSS makes it possible to place the whole installation directory of a product in the DCSS, and place only the directories or files of the local instance (containing files that are updated at runtime) in local file system (if supported). Alternatively, a soft link might be used to connect to the local directory and the DCSS. The use of soft links works in both directions, because Linux resolves soft links using file names. This approach simplifies the whole administration significantly, and eliminates the dependency of the local instance from the version of the software package.

Sharing read-only data

A database workload with read-only data is used to compare the throughput using a large DCSS with the throughput using disk devices.

Use as swap device

A guest can use a DCSS in exclusive write access as swap device. This has the advantage that the guest stays in the Start Interpretive Execution (SIE) instruction when swapping to the DCSS.

A performance comparison of a VDISK, a DCSS, or a DASD as swap device with single guests will help when planning system setup.

This paper shows the impact on performance of the total system using the scenarios described above, with or without a DCSS. These results include the consideration of how many guests are needed to provide a benefit in total memory consumption.

Summary

Two types of DCSS (SN and SR) are described. Their differences are discussed in detail.

When initially saving a new DCSS from a CMS user, the two types (SN and SR) behave very differently. These are the key differences:

- While the DCSS of type SN is saved almost immediately, the DCSS of type SR takes a considerable amount of time, depending on its size (approximately 12 seconds plus 81 seconds for each GB of memory).
- The DCSS of type SN does not save any data to the disk. All data written to the DCSS are lost when the last Linux user removes the DCSS, or when there is a system restart.
- The data of a DCSS of type SR are persistent, because this DCSS is saved to the spool area. A disk setup with a high disk I/O bandwidth is strongly recommended for the spool area. An example is multiple dedicated disks from several ranks, see: http://public.dhe.ibm.com/software/dw/linux390/perf/disk_performance_optimizing.pdf
- This observation is the same for all operations such as loading, saving, or writing from a Linux guest. Because there are always disk transfers involved, these types of operations with a DCSS of type SR are generally slower than with a DCSS of type SN. Fortunately, saving or loading are typically done rarely, and loading is faster than saving (approximately 12 seconds plus 63 seconds per GB of memory). The time to fill the whole DCSS of type SR with data is much faster than the time needed for loading it (approximately 12 seconds plus 11 seconds per GB of memory).

The shorter access time and the lack of persistence of a DCSS of type SN makes it useful for temporary data, such as a swap device. As a swap device, the DCSS use is exclusive to each guest. Comparing VDISK and DCSS, as memory-based devices, with a physical DASD device in configurations where Linux is swapping, shows that using the DCSS of type SN provides better application performance than VDISK. The use of a VDISK cannot be recommended, especially with many guests.

The test results indicate that the best performance is obtained when the memory is used to increase the guest size instead of the DCSS size. This result is due to reduced requirements from Linux to swap memory, and therefore reduced effort for memory management. This configuration (guest size is increased by the size of the DCSS) seems to be a better use of the memory than using it for a fast swapping device.

The use of the VDISK or DCSS might provide an advantage when used as a hot standby swap resource, which is used only for short workload peaks. With the use of the hot standby, the memory is most of the time not allocated from z/VM or main memory. The additional memory of a guest is used most of the time, especially when doing file I/O. For the case of

permanent Linux swapping activity (not a short workload peak), increasing the guest size is a better way to improve the performance.

When trying to determine the memory savings using a DCSS, it was found that only 400 MB from the 6 GB memory of the DCSS is loaded into real memory. Only the pages used are loaded, not the whole DCSS. The pages needed at runtime (and therefore needed to stay in real memory) total most of the time approximately 10 MB in size, which is relatively small.

The transfer rates between DCSS and XSTOR show that the pages that are most often needed stay in real memory. To identify the impact of the memory savings using a DCSS, a comparison was done without DCSS and with increased guests size to compensate for the additional memory needed for the WebSphere® binaries. Adding 10 MB, as used from the DCSS in the test before, to each guest produces a suboptimal result. This is due of the increase in memory pressure inside z/VM (caused by the loss of the memory given to the guests).

Adding only 5 MB to each guest results in the best performance without the DCSS, but this scenario is still slower than the throughput achieved with the DCSS. The conclusion is that with five or more WebSphere Application Server guests that include DB2®, running the binaries with the execute-in-place option from the DCSS is recommended. The memory savings is a minimum of approximately 10 MB per guest, which would result in a total savings of approximately 1 GB for 100 guests.

A really great advantage of the DCSS is seen when using it to share read-only data, which is highly recommended. Using a correspondingly sized minidisk cache improves the throughput by 15% compared to minidisk without the cache. Placing the data into a DCSS improves the throughput dramatically, by almost a factor of 10.

Test system environment and configuration

The test system for DCSS performance testing was an IBM System z10® with necessary software components, and other systems to generate the workload.

Hardware configuration

The hardware configuration consists of an IBM System z10 using LPAR, and Linux servers to generate the workload.

The host hardware is listed in Table 1.

Table 1. Hardware configuration of the host

System	Operating System	Number of systems	Number of processors	Memory in GB	XSTOR	Network	Architecture
z/VM LPAR	z/VM 5.4	1	10	Between 3326 MB and 25 GB	2 GB	One OSA (1 Gb)	z10™ LPAR
Linux Servers	SLES 11	1 - 10	1 or 2	650 MB or 768 MB		Shared OSA	z/VM guest

Workload generating clients

The workload generating clients are two 4-way systems running Linux:

- One for creating the DayTrader workload
- One for creating the Swingbench or Bookstore workload

Software configuration

The software for the DCSS performance analysis consists of: z/VM, Linux, WebSphere, DB2, and three workload programs.

The software configuration is described in Table 2.

Table 2. Software configuration

Product	Version and Release	Comments
DayTrader	2.0 ee5, r343	Use with normal database
Bookstore	n/a	Use with read-only database
WebSphere	7.1 (31-bit version)	▪ Latest fix pack ▪ The 31-bit version is used because a system with less than 2 GB of memory does not require 64-bit
DB2 for Linux	9.7	Latest fix pack
z/VM	5.4 RSU 0902	Order number UM97540
Novell SUSE	SLES 11	Latest update
Swingbench	2.3.0.422	Use with read-only database

Test environment

The test environment for the DCSS performance analysis consisted of several guests running WebSphere Application Server and DB2. These guests received work from two workload generator systems, running thee workload applications.

The test environment is illustrated in Figure 1.

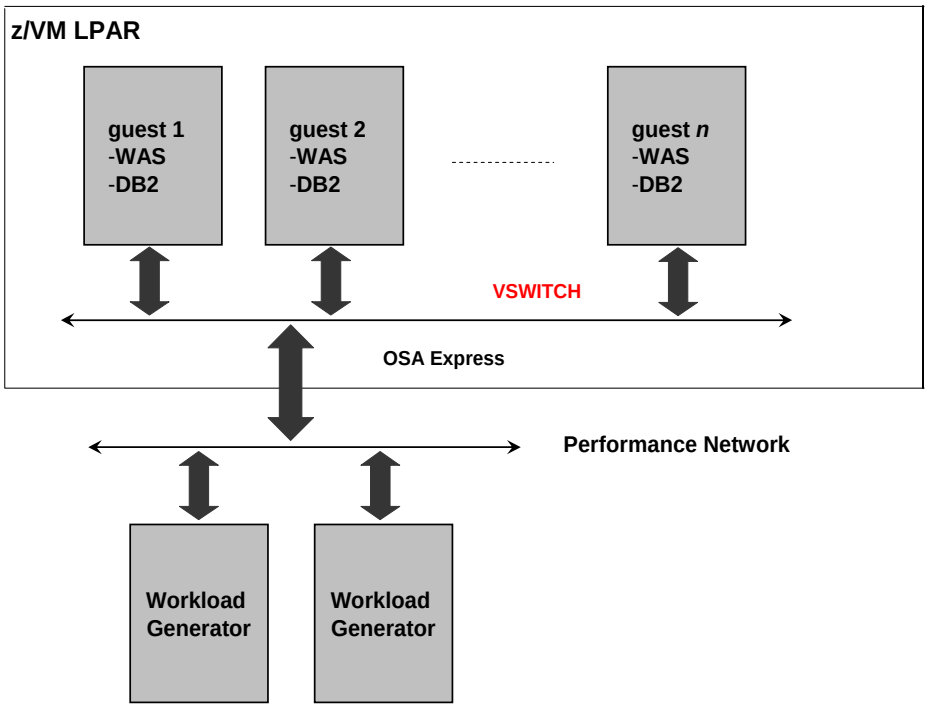

Figure 1:

Network environment

The network environment consists of:

- An external interface, a 1 Gb Ethernet card
- Internal connectivity provided by use of a VSWITCH

DCSS and test environment setup

The test environment for the DCSS performance analysis was modeled after several existing configurations.

The following references were used to set up the test environments:

- How to use Execute-in-Place Technology with Linux on z/VM
- How to - Share a WebSphere Application Server V7 installation among many Linux for IBM System z systems

Planning a DCSS

When planning for DCSS creation, the DCSS size and address ranges must be chosen carefully.

Table 3 shows the sizes of the DCSSs used in this paper:

Table 3. DCSS configuration: Allocated address ranges

DCSS name	Total size	Segment addresses	Description
ORADB1-5	10 GB	22FC00 - 4AF6FF	Read-only database
DCSSOPT1-3	6 GB	4AF700 - 62F3FF	Shared binaries
SWAPPING	1 GB	62F400 - 66F3FF	Swapping space
DCSS1G	1 GB	20000 - 5FFFF	Used for measuring DCSS load times
DCSS2G	2 GB	20000 - 9FEFF	Used for measuring DCSS load times
DCSS4G1-2	4 GB	20000 - 11FDFF	Used for measuring DCSS load times
DCSS8G1-4	8 GB	20000 - 21FBFF	Used for measuring DCSS load times
DCSS16G1-8	16 GB	20000 - 41F7FF	Used for measuring DCSS load times

Figure 2 illustrates the memory allocation for the Linux system, and how some memory segments use a DCSS.

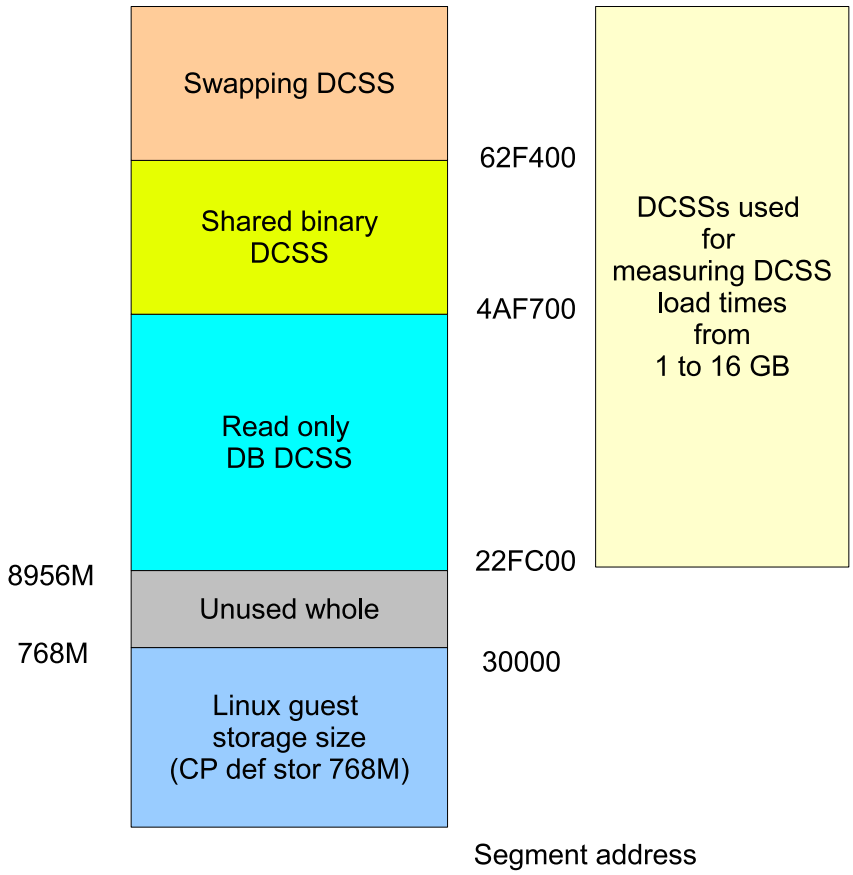

Figure 2. Linux memory footprint

Prerequisites for defining a DCSS

Before defining a DCSS, it is necessary to ensure that users and system space are set up properly.

A DCSS must be defined in a contiguous address range. Make sure that the CMS user storage size is large enough to map the address range of the DCSS, and that this storage remains in a contiguous block.

To create a DCSS, complete these steps:

1. Make sure that the CMS user ID is defined with Class E privileges, by doing one of these two tasks:
   - Issue this command from the user ID maint:

- SET PRIVCLASS userid +E
- Add class E to the CLASS string in the CMS user ID's directory

```
Example:
CLASS z16axcb25ge
```

This string specifies that the user ID can enter commands with privilege classes `Z, 1, 6, A, X, C, B, 2, 5, G and E.`

2 For a DCSS of type SR, ensure that there is sufficient spool space on the z/VM system to contain any DCSSs that are created. A recommendation is to add to the normal size of the spool space, twice the total size of the DCSSs to be created.

3 Ensure that the CMS user ID has sufficient defined storage to map the DCSS (CP def stor command). For example, if the DCSS is 1 GB in size, then the user must define storage at least 1 GB greater than the starting address of the DCSS (DCSS starting segment address plus X'60000' for a DCSS of 1 GB in size).

4 Ensure that the CMS user ID has sufficient storage to save the DCSS. This means that the user must have storage that maps the highest address specified in the DEFSEG command plus the size of the DCSS.

For more information about DEFSEG and SAVESEG, see:
z/VM: CP Commands and Utilities Reference

Defining a DCSS

To define a DCSS, the CP DEFSEG and SAVESEG commands are invoked from a CMS user ID. Both these commands are Class E CP commands.

To define a DCSS, on the CMS user ID:

1. Define the DCSS using the DEFSEG command:

   This example is for a 1 GB DCSS of type SN starting at the 512 MB boundary

   ```
   defseg DCSS1G  20000-5FFFF sn
   ```

   A skeleton (class S) system data file for the saved segment is created.

For more information, see Appendix. The CP DEFSEG and SAVESEG commands.

2. Save the segment with the SAVESEG command:

   ```
   saveseg  DCSS1G
   ```

This command produces a response from z/VM indicating that the DCSS was saved to the spool, similar to this message:

```
00:HCPNSS440I Saved segment DCSS1G was successfully saved in
file ID 0400.
```

Now the saved segment can be accessed. The SAVESEG command changes a skeleton file to an active (class A or R) file.

3. Issue this command to check the results of the SAVEDCSS exec:

   ```
   q nss name map dcssname
   ```

DCSS type: SN versus SR

There are two types of shared segment for Linux use. They are SN (shared read/write access) and SR (shared read-only access).

Table 4 lists some trade-offs for DCSS type SN versus SR.

Table 4. Comparison of DCSS types SN and SR

Consideration	DCSS type	
	SN: Shared R/W	SR: Shared R/O
Initial elapsed time to populate the DCSS with the files to be shared	Faster because no DASD I/O is necessary	Slower because DASD I/O is required
File system loaded into DCSS is written to z/VM spool	No	Yes
Spool processing for DCSS can delay other spool activity	No	Yes
Persistent after system restart	No	Yes

Defining a DCSS for measuring DCSS load times

Two types of DCSS, SN and SR were created, each having these sizes: 1 GB, 2 GB, 4 GB, 8 GB, and 16 GB.

For a DCSS larger than 2 GB, several 2 GB DCSS segments were created and concatenated together on the Linux system to create a single DCSS of the desired size. See Examples for the commands used for these DCSS definitions.

Creating a DCSS of type SN

These steps are used to create a DCSS of type SN. Commands must be issued on both the z/VM and Linux systems.

On a CMS user ID

1. Ensure that the user has sufficient storage to save the `DCSS`. This means that the user must have storage that maps the highest address specified in the `DEFSEG` command, plus the `DCSS` size.

2. Define a 1 GB, 2 GB, 4 GB, 8 GB, or 16 GB DCSS with the `DEFSEG` command

   This example is for a 4 GB DCSS:

   ```
   defseg DCSS4G1 20000-6FFFF sn
   defseg DCSS4G2 70000-BFFFF sn
   ```

3. Save the segment with the SAVESEG command

   This example is for a 4 GB DCSS:

   ```
   saveseg DCSS4G1
   saveseg DCSS4G2
   ```

   The system produces a response indicating that the DCSS was saved to the spool for each DCSS segment, similar to this message:

   ```
   00: HCPNSS440I Saved segment DCSS1G1 was successfully saved
    in
        file ID 0400.
   ```

4. Check the results of the `SAVESEG` command by issuing this command:

   ```
   q nss name map dcssname
   ```

   See Appendix. The CP DEFSEG and SAVESEG commands for a list of the `DEFSEG` commands used.

On the Linux guest

1. Allocate 512 MB of storage by issuing this command:
   ```
   def stor 512M
   ```

2. Restart Linux

3. If the mem parameter in file /etc/zipl.conf has not already been adjusted, complete these steps:

   a. Set the mem parameter to the size of the guest memory, that is 512 MB plus the size of the DCSS.

As an example, for a 1 GB DCSS, add mem=1536M to the parameters line.

b. Issue the `zipl` command.

c. Restart Linux.

4. Load the DCSS block device driver, by issuing this command:
   `modprobe dcssblk`

5. Identify the major device number for the `dcssblk` device and create a device node for one or more `DCSS` devices, by issuing this command:

```
6.       cat /proc/devices
         mknod /dev/dcssblk0 b 252 0
```

This example used minor number 0 because the dcsblk0 device did not exist. However, the minor number chosen must be unique.

7. Load the `DCSS:`

```
echo "DCSS1G" > /sys/devices/dcssblk/add
```

Loading a `DCSS` creates a subdirectory with the name of the `DCSS` within the /sys/devices/dcssblk directory. This subdirectory contains three attributes and a symbolic link named block. The symbolic link points to the block device that corresponds to the `DCSS.`

This ls command illustrates the symbolic link:

```
lnx001:~ # ls -l /sys/devices/dcssblk/DCSS1G
total 0
drwxr-xr-x 3 root root    0 Aug  5 10:15 block
-rw------- 1 root root 4096 Aug  5 10:18 save
-r-------- 1 root root 4096 Aug  5 10:18 seglist
-rw------- 1 root root 4096 Aug  5 10:18 shared
-rw-r--r-- 1 root root 4096 Aug  5 10:18 uevent

lnx001:~ # ls /sys/devices/dcssblk/DCSS1G/block/dcssblk0
bdi         dev     holders  removable  size    stat       uevent
capability  device  range    ro         slaves  subsystem
```

The larger DCSSs were defined with this command sequence:

```
echo "DCSS2G" > /sys/devices/dcssblk/add
echo DCSS4G1:DCSS4G2 > /sys/devices/dcssblk/add
echo DCSS8G1:DCSS8G2:DCS8G3:DCS8G4 > /sys/devices/dcssblk/add
echo DCSS16G1:DCSS16G2:DCS16G3:DCS16G4:
     DCSS16G5:DCSS16G6:DCS16G7:DCS16G8 > /sys/devices/dcssblk/add
```

8. Create an ext2 file system on the `DCSS`, by issuing this command:

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

9. Mount the DCSS by issuing this command:

```
mount -t ext2 -o xip /dev/dcssblk0 /mnt
```

Creating a DCSS of type SR

These steps are used to create a DCSS of type SR. Commands must be issued on both the z/VM and Linux systems.

On a CMS user ID

Perform the same steps as described in Creating a DCSS of type SN, except issue the `DEFSEG` command with type SR and option `LOADNSHR`.

On the Linux guest

1. Allocate 512 MB of storage by issuing this command:
   ```
   def stor 512M
   ```
2. Restart Linux
3. Edit the `file /etc/zipl.conf` (do this only once) and make this change:
   - For a 1 GB DCSS, add `"mem=1536M"` to the parameters line. For a larger `DCSS`, the mem= parameter must be increased. It is set to the size of the guest memory `(512 MB` plus the size of the `DCSS)`
4. Issue the `zipl` command.
5. Restart Linux
6. Issue this command to load the DCSS block device driver:
   ```
   modprobe dcssblk
   ```
7. Issue this command to get the major device number for the `dcssblk` device:
   ```
   cat /proc/devices
   ```
8. Issue this command to create a device node for one or more `DCSS` devices:
   ```
   mknod /dev/dcssblk0 b 252 0
   ```
   This example uses minor number `0` because the `dcsblk0` device did not exist. However, the minor number chosen must be unique.
9. Issue this command to load the DCSS:
   ```
   echo "DCSS1G" > /sys/devices/dcssblk/add
   ```

Loading a DCSS creates a subdirectory with the name of the `DCSS` within the `/sys/devices/dcssblk` directory. This subdirectory contains three attributes and a symbolic link, named block. The symbolic link points to the block device that corresponds to the `DCSS`.

This `ls` command illustrates the symbolic link:

```
lnx001:~ # ls -l /sys/devices/dcssblk/DCSS1G
total 0
drwxr-xr-x 3 root root    0 Aug  5 10:15 block
-rw------- 1 root root 4096 Aug  5 10:18 save
-r-------- 1 root root 4096 Aug  5 10:18 seglist
-rw------- 1 root root 4096 Aug  5 10:18 shared
-rw-r--r-- 1 root root 4096 Aug  5 10:18 uevent
lnx001:~ # ls /sys/devices/dcssblk/DCSS1G/block/dcssblk0
bdi         dev     holders  removable  size    stat       uevent
capability  device  range    ro         slaves  subsystem
lnx001:~ #
```

The larger DCSSs were defined with this command sequence:

```
echo "DCSS2G" > /sys/devices/dcssblk/add
echo DCSS4G1:DCSS4G2 > /sys/devices/dcssblk/add
echo DCSS8G1:DCSS8G2:DCS8G3:DCS8G4 > /sys/devices/dcssblk/add
echo DCSS16G1:DCSS16G2:DCS16G3:DCS16G4:
     DCSS16G5:DCSS16G6:DCS16G7:DCS16G8 > /sys/devices/dcssblk/add
```

10. Make the DCSS writable by setting the shared value to exclusive use. Because this DCSS is type SR, it is initially attached by the Linux kernel as read-only.

```
11.  echo 0 > /sys/devices/dcssblk/DCSS1G/shared
12.  echo 0 > /sys/devices/dcssblk/DCSS2G/shared
13.  echo 0 > /sys/devices/dcssblk/DCSS4G1/shared
14.  echo 0 > /sys/devices/dcssblk/DCSS8G1/shared
     echo 0 > /sys/devices/dcssblk/DCSS16G1/shared
```

15. Make an ext2 file system on the DCSS by issuing this command:
    `mke2fs -b 4096 /dev/dcssblk0`

16. Mount the DCSS by issuing this command:
    `mount -t ext2 -o xip /dev/dcssblk0 /mnt`

17. Copy data to the DCSS.

18. Save the DCSS. The DCSS is changed only in memory. To move the changes into the DCSS residing on the spool file, issue these commands:

```
19.  echo 1 > /sys/devices/dcssblk/DCSS1G/save
20.  echo 1 > /sys/devices/dcssblk/DCSS2G/save
21.  echo 1 > /sys/devices/dcssblk/DCSS4G1/save
22.  echo 1 > /sys/devices/dcssblk/DCSS8G1/save
     echo 1 > /sys/devices/dcssblk/DCSS16G1/save
```

23. Unmount the DCSS by issuing this command:
    umount /mnt

24. Remove the DCSS to cause the newly saved DCSS to become active in the spool, by issuing these commands:

```
25.  echo "DCSS1G" > /sys/devices/dcssblk/remove
26.  echo "DCSS2G" > /sys/devices/dcssblk/remove
27.  echo "DCSS4G1" > /sys/devices/dcssblk/remove
28.  echo "DCSS8G1" > /sys/devices/dcssblk/remove
     echo "DCSS16G1" > /sys/devices/dcssblk/remove
```

Defining a DCSS for swapping

These steps are used to define a DCSS to be used as a swapping device. This DCSS has a size of 1 GB.

On a CMS user ID

1. Make sure that the user ID creating the DCSS has the correct privilege class (example: set `privclass lnx00090 +e`) and sufficient storage.

2. Define a 1 GB DCSS named SWAPPING using this DEFSEG command:
   `defseg swapping 62f400-62f400 ew 62f401-66f3ff en`

   The first page (62F400) is writable so that the swap signature can be written to it. The rest of the DCSS is used for the swap data, so it never has to be saved.

3. In this test, a DCSS of type SN was also defined for swapping:
   `defseg swapping  62f400-66f3ff sn`

   Note: Even though the DCSS is of type shared, it is used here as a private DCSS for one guest.

4. Save the segment by issuing this SAVESEG command:
   `saveseg swapping`

The system issues a response indicating that the DCSS was saved to the spool, similar to this message:

```
00: HCPNSS440I Saved segment SWAPPING was successfully saved
    in fileid 0398.
```

5. Query the new DCSS by issuing the command: `q nss name swapping map`

This is the output for the DCSS of type EW/EN:

```
00: FILE FILENAME FILETYPE BEGPAG        ENDPAG        TYPE CL #USERS
00: 0398 SWAPPING DCSSG 000000062F400 000000062F400 EW A  00000
00:                     000000062F401 000000066F3FF EN
```

This is the output for the DCSS of type SN:

```
00: FILE FILENAME FILETYPE BEGPAG        ENDPAG        TYPE CL #USERS
00: 0398 SWAPPING DCSSG 000000062F400 000000066F3FF SN A  00000
```

The DCSS for the Linux swap spaces is now defined. Any Linux virtual machine can use it.

On the Linux guest

1. Attach and configure the DCSS to use the swap space by issuing these commands:

```
2. modprobe dcssblk
3. mknod /dev/dcssblk0 b 252 0
4. echo SWAPPING > /sys/devices/dcssblk/add
5. mkswap /dev/dcssblk0
   swapon /dev/dcssblk0
```

6. To list the swap devices, issue this command:
   `swapon -s`

The output is similar to this message:

```
Filename                        Type            Size    Used
  Priority
/dev/dcssblk0                   partition       1048568 0       -1
```

To set off other swap devices, issue the `swapoff` command.

Defining a VDISK for swapping

These steps are used to define a VDISK as a swapping device for the Linux guest.

Note: The `SET VDISK` command controls the maximum amount of host real storage that is available for allocation as virtual disks in storage. Using virtual disks in storage increases the load on system paging, so limits should be chosen in proportion to the availability of paging space.

1. Set the total resource available for allocating virtual disks in storage on the system, by issuing this command:
   `CP set vdisk syslim 1g`

2. Set the maximum resource available for virtual disks in storage created by a single user who issues the DEFINE command:
   `CP set vdisk userlim infinite`

3. Define the VDISK (in 512 byte blocks) by issuing these commands:

```
4.  def vfb-512 2000 2097152
5.  DASD 2000 DEFINED
6.  Ready; T=0.01/0.01 12:01:43
7.
8.  def 2000 29f
9.  DASD 029F DEFINED
10. Ready; T=0.01/0.01 12:02:00
11.
12. q v 29f
13. DASD 029F 9336 (VDSK) R/W    2097152 BLK ON DASD  VDSK SUBCHANNEL
    = 001F
    Ready; T=0.01/0.01 12:02:15
```

14. Issue the mkswap command to set up the VDISK as swap space:

```
15. lnx001:~ # mkswap /dev/dasdw1
16. Setting up swapspace version 1, size = 1048568 KiB
    no label, UUID=78381348-4148-42be-83f4-8363eeef6234
```

17. Issue the `swapon` command to set the VDISK as the only swap device on the system:

```
18. lnx001:~ # swapon /dev/dasdw1
19. lnx001:~ # swapon -s
20. Filename                Type            Size    Used    Priority
21. /dev/dasda2                      partition       1130392 231380  -1
22. /dev/dasdw1                      partition       1048564 0       -2
23. lnx001:~ # swapoff /dev/dasda2
24. lnx001:~ # swapon -s
25. Filename                Type            Size    Used    Priority
    /dev/dasdw1                       partition       1048564 277128 -1
```

An alternative is to update the fstab entries as required.

Defining a VDISK using the SWAPGEN exec

An alternative to Steps 1 - 4 of the previous procedure is to define the VDISK swap space by using the `SWAPGEN exec`.

The exec syntax is:
`SWAPGEN vdev #blocks <(<options> <)> >`

where:
vdev

Is a virtual device address
#blocks

Is a decimal number of 512-byte blocks. The minimum is 24 (for FBA option) or 32 (for DIAG option).

Options are:
DIAG

Use DIAG I/O (requires Linux DIAG driver). This is the default.
FBA

Use FBA driver instead of DIAG. This option requires the RXDASD package, which can be downloaded from the IBM VM download page at:
http://www.vm.ibm.com/download/packages

REUSE
Use existing device at vdev. **WARNING**: Using this option will destroy any data on device vdev.

An example is:
```
ex SWAPGEN 300 2097152 ,
```

Running a WebSphere Application Server and a DB2 database from a shared DCSS

These steps set up a WebSphere Application Server (WAS) and DB2 database so that they can use a shared DCSS.

Defining a DCSS for WebSphere and DB2

To *define* a DCSS that holds the binaries of WebSphere and DB2 (or of another typical database of 6 GB in size) consisting of three 2 GB segments, complete these steps:

1. Install the software product on a normal disk.
2. Copy the required installed directories to the DCSS.
3. Mount the DCSS in execute-in-place (XIP) mode.
4. On each Linux guest, establish symbolic links that point to the directories of the software products loaded in the DCSS.
5. On each Linux guest, store or generate instance-specific information.

In a production environment, there is a need to install patches and fix packs on software that resides in the DCSS. Other than for cloned Linux systems that automatically pick up changes, a DCSS must be updated explicitly with patches or fixes, or a new DCSS must be created, which enables different systems to operate at different software levels.

Updating a DCSS for WebSphere and DB2

To *update* a DCSS, complete these steps:

On a CMS user ID

1. Define a 6 GB DCSS with the DEFDCSS exec
   DEFDCSS DCSSOPT 4AF700 3 2047 SR
2. Save the segment with the SAVEDCSS exec
   SAVEDCSS DCSSOPT 3

On the Linux guest

1. Install the software packages on the DASD.
2. Copy the required installed directories to the DCSS.
3. Save the DCSS
4. On each Linux guest, establish symbolic links from the original installation directories in /opt, which point to the directories of the software products loaded in the DCSS.
5. To create the file system and copy data into it, the DCSS must be in exclusive writable mode.
6. Attach the DCSS by issuing these commands:

```
7.  modprobe dcssblk
8.  mknod /dev/dcssblk0 b 252 0
9.  echo DCSSOPT1:DCSSOPT2:DCSSOPT3  > /sys/devices/dcssblk/add
10. echo 0 > /sys/devices/dcssblk/DCSSOPT1/shared
11. mke2fs -b 4096 /dev/dcssblk0
    mount  /dev/dcssblk0 /optdcss/
```

12. Copy the required files for WebSphere and DB2 to the DCSS, as explained in Setting up a WebSphere Application Server to use a DCSS and Setting up a DB2 server to use a DCSS.

13. Save the DCSS by issuing this command. This command schedules the current storage copy of the DCSS to be written to the z/VM spool.
    ```
    echo 1 > /sys/devices/dcssblk/DCSSOPT1/saved
    ```
    The changes to the DCSS are now saved in the z/VM spool with a class of "P" for Pending.

    When the last z/VM guest stops accessing the old version of the DCSS, the old version in memory is discarded. Each guest that opens the DCSS accesses the updated copy.

14. To *release* the DCSS from the Linux guest, unmount the DCSS and then remove it:
    ```
    echo DCSSOPT1 >  /sys/devices/dcssblk/remove
    ```
    The z/VM system updates the spool files, deletes the old spool file of class "A", updates the new spool file of class "P", and changes its class to "A" for Active. The next access of the DCSS sees the changed version.

For more information about how to set up the Linux system for using a DCSS to hold read-only copies of software binaries, see:

- Sharing and Maintaining SLES 11 Linux under z/VM using DCSSs and an NSS
- How to - Share a WebSphere Application Server V7 installation among many Linux for IBM System z systems

Setting up a WebSphere Application Server to use a DCSS

These steps are used to configure a WebSphere Application Server (WAS) so that it can use a DCSS for the WebSphere executable code.

1. Install WebSphere according to the installation instructions with one exception: Skip the profile creation step and create the profile in a separate directory structure before the WebSphere binaries are copied to the DCSS.

2. Mount the DCSS by issuing these commands:

```
3. modprobe dcssblk
4. mknod /dev/dcssblk0 b 252 0
```

5. `echo DCSSOPT1:DCSSOPT2:DCSSOPT3 > /sys/devices/dcssblk/add`
   `mount -t ext2 -o xip /dev/dcssblk0 /optdcss/`
6. Place the WebSphere installation in directory `/opt/IBM/WebSphere/AppServer`.
7. Copy the files from directory `/opt/IBM/WebSphere/AppServer` to the DCSS in `/optdcss/WebSphere/AppServer/`.
8. Link the directory `/opt/IBM/WebSphere/AppServer` to `/optdcss/WebSphere/AppServer/`.

This ls command output illustrates the contents of directory `/optdcss/WebSphere/AppServer/`:

```
ls /optdcss/WebSphere/AppServer/
Scheduler     features            logs              systemApps
UDDIReg       firststeps          lost+found        temp
bin           installableApps     optionalLibraries uninstall
cimrepos      installedConnectors plugins           universalDriver
configuration instutils           profileTemplates  util
deploytool    java                properties        web
derby         lafiles             runtimes
dev           lib                 sar2war_tool
etc           links               scriptLibraries
```

9. Create the WebSphere profiles on DASD at `/opt/wasprofile/` following the instructions on creating WebSphere profiles from:

   How to - Share a WebSphere Application Server V7 installation among many Linux for IBM System z systems

Setting up a DB2 server to use a DCSS

These steps are used to set up DB2 so that a DCSS can be used for the DB2 executable code.

1. Install the DB2 server on DASD following the normal installation instructions, however skip the DB2 instance creation.
2. Copy the files located in directory `/opt/ibm/db2` to the DCSS directory `/optdcss/db2`.
3. Symbolically link `/opt/db2` to `/optdcss/db2`.
4. Set up and mount the DCSS.
5. Create the DB2 instance named **db2inst1** using the DB2 command-line functions.

This ls command illustrated the contents of the `/optdcss/db2/V9.7/` directory:

```
ls  /optdcss/db2/V9.7/
```

```
.metadata  bnd    desktop  include  java    map    security32
       Readme     cfg    doc      infopop  lib32   misc
security64
       adm        conv   function install  lib64   msg
tivready
       adsm       das    gskit    instance license properties tools
       bin        dasfcn ha       itma     logs    samples
```

Workload description

The workload used to test large DCSS configurations used three applications, named Swingbench, DayTrader, and Bookstore.

Swingbench

Swingbench is a Java™-based application that generates a workload used to run stress tests on a relational database. Swingbench comes with several predefined benchmarks, but also has the capability for the user to define their own customized benchmarks.

Swingbench has these components:

- The load generator
- The coordinator
- The cluster overview

The test scenarios described in this paper used Swingbench to generate a workload and record statistics for later analysis. In the test scenarios, the Swingbench data base was used for read-only transactions.

For more information about Swingbench, see:
http://dominicgiles.com/swingbench.html

DayTrader

DayTrader is a benchmark application that simulates an online stock trading system.

DayTrader was originally developed by IBM with the name 'Trade Performance Benchmark Sample'. In 2005, DayTrader was donated by IBM to the Apache Geronimo community. A typical user of DayTrader performs these tasks:

- Log in the DayTrader user interface.
- View the user's stock portfolio.
- Find current stock prices.
- Buy or sell shares of stock.
- Log off the DayTrader user interface.

With the aid of a Web-based load driver such as Mercury LoadRunner, Rational® Performance Tester, or Apache JMeter, the real-world workload provided by DayTrader can be used to measure and compare the performance of Java Platform, Enterprise Edition (Java EE) application servers offered by various vendors. In addition to the full workload, DayTrader also contains a set of primitives used for functional and performance testing of various Java EE components and common design patterns.

DayTrader requires a read/write data base, and was used to provide a transactional workload on WebSphere and DB2.

For more information about DayTrader, see:
https://cwiki.apache.org/GMOxDOC20/daytrader.html

Bookstore

Bookstore is an IBM internally developed application used to simulate an online bookstore.

Bookstore is used to search for books by title and author, add books to a shopping cart, purchase the contents of a shopping cart, get the status on a previously placed order, and other functions. The database supporting the Bookstore application contains information about titles, authors, stores, customers, and orders.

The test scenarios described in this paper used only the search features of the Bookstore application.

Results

Tests were run altering DCSS setup and tuning parameters to see how system performance is affected. Different DCSS sizes were used.

DCSS access and load times

This test case measures the time to load a DCSS into z/VM memory, by writing data to it. DCSS types SN and SR, with sizes of 1 GB, 2 GB, 4 GB, 8 GB, and 16 GB were tested.

For a description of the setup steps, see Creating a DCSS of type SN and Creating a DCSS of type SR.

DCSS of type SN

The time to fill a DCSS of type SN with random data is measured and analyzed.

This Linux command was used to create a 256 MB file of random data:

```
dd if=/dev/urandom of=/mnt2/testcase0-data bs=268435456 count=1
```

Then this data was copied to the DCSS device repetitively, until the device was full.

The Linux `time` command was used to record the time taken by the copy operation.

A DCSS of type SN is not saved by z/VM to the spool space. The data copied into the DCSS remain there until the last Linux guest releases and removes the DCSS. Then, when no guests are accessing the DCSS, the data in the DCSS is removed.

The save time on CP and the initial access time from Linux are negligible, because this type of DCSS does not write the data into the spool file.

Figure 3 shows the times to copy data into a DCSS for various DCSS sizes.

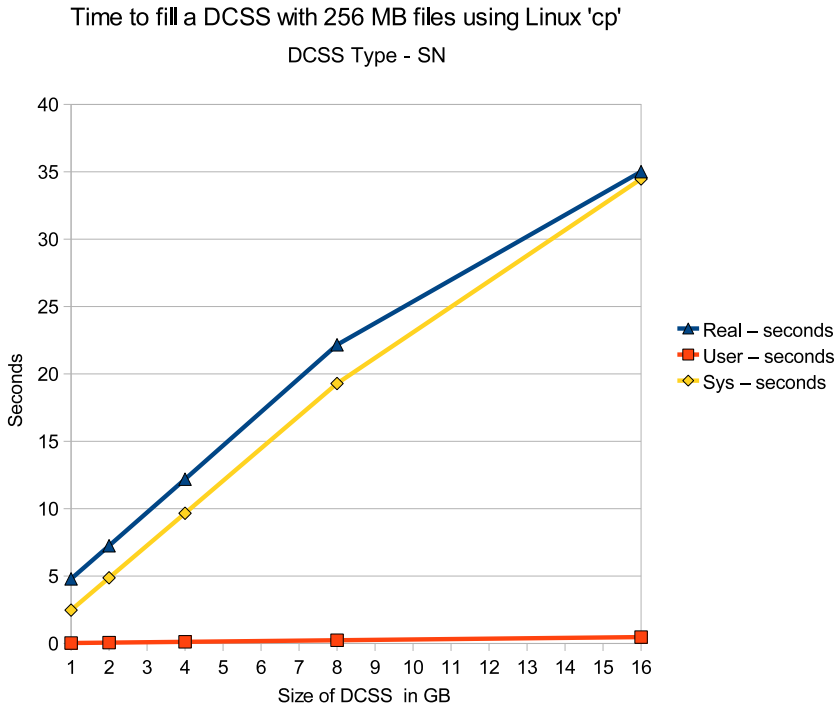

Figure 3. Time to fill a DCSS of type SN

Observations
The time to fill a DCSS of type SN is a linear function of the DCSS size.

Most of the time for this operation is used for the *system* in kernel mode. The *real time*, which is the elapsed time to run this operation, is slightly higher than the sum of user and system time. This difference reflects the time that the command is processed on z/VM.

Conclusions
The DCSS of type SN has little overhead on CP to create and save date. The initial access of the DCSS on Linux is very fast, even for a 16 GB DCSS.

However, this type of DCSS does not save data in the spool, and data written to the DCSS are lost when the last Linux user removes the DCSS.

DCSS of type SR

The time to fill, save, and reload a DCSS of type SR is recorded and analyzed.

This Linux command was used to create a 256 MB file of random data:

```
dd if=/dev/urandom of=/mnt2/testcase0-data bs=268435456 count=1
```

Then this data was copied to the DCSS device repetitively, until the device was full.

When a DCSS of type SR is saved, its content is written to the spool space. When pages are loaded from a DCSS of type SR, the required pages are read from the spool space.

The times for all intermediate steps, from creating the DCSS until finally accessing the filled and saved DCSS, was measured and recorded.

The time is recorded for these specific steps:

- Time to initially save the DCSS from the CMS user
- Time to initially access the DCSS from the Linux guest
- Time to copy data to the DCSS until it was full
- Time to save the populated DCSS in the z/VM spool space
- Time to re-access the DCSS

The Linux `time` command was used to record the time of each of the above steps.

Time to save the DCSS in CP

The system time returned in the CP command response is used to measure the amount of time that it takes for the SAVESEG command to complete for a DCSS of type SR and various different sizes.

Figure 4 shows the time that it takes for the SAVESEG command to complete for various DCSS sizes.

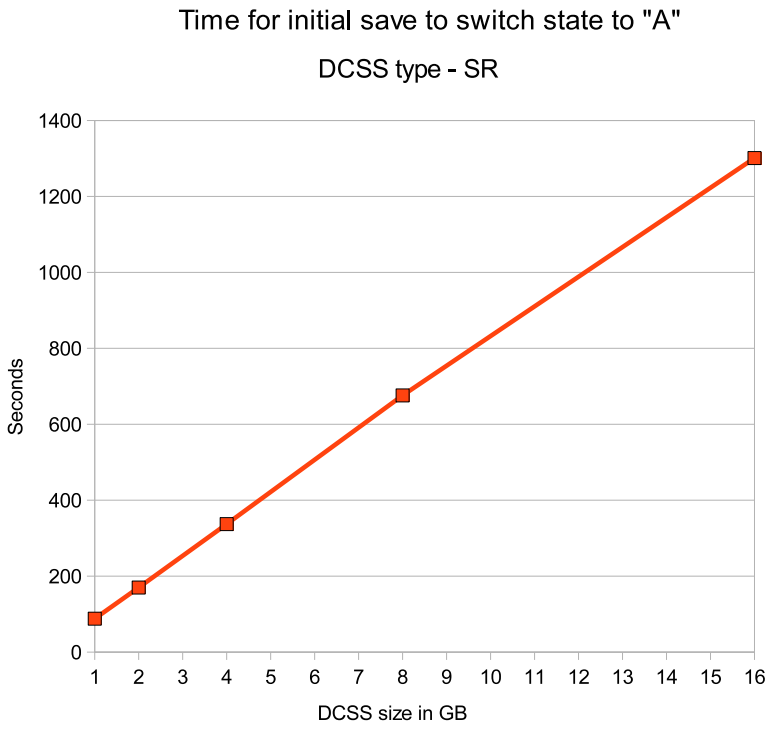

Figure 4. Time for the SAVESEG command to complete, for a DCSS of type SR

Observation

The time to complete the SAVESEG command is a linear function of the DCSS size. All DCSS pages are written to the spool file, even when they are empty.

Conclusions

The save times reflect the fact that the DCSS pages are written to the spool space on DASD. For a large DCSS, this can become a significant amount of time (23 minutes for a 16 GB DCSS).

When using a DCSS of type SR, consider using devices with the best possible I/O bandwidth for the spool area.

Time to access the DCSS on Linux

The time for the first Linux user to access a DCSS of type SR is measures for various DCSS sizes using the command: `echo` *`DCSS name`* `> /sys/devices/dcssblk/add`.

Figure 5 shows the time to complete the DCSS `add` function for various DCSS sizes.

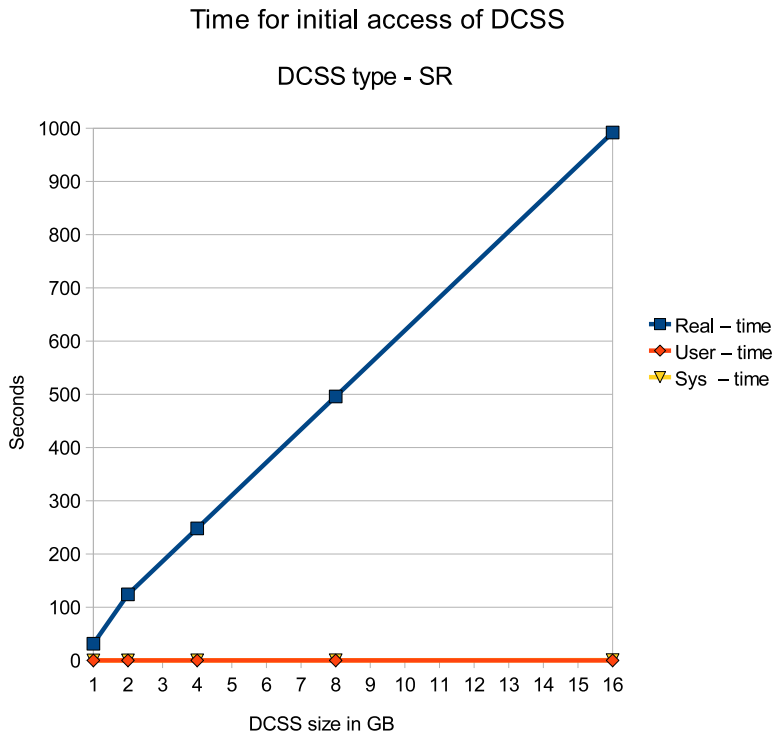

Figure 5. Time to complete the Linux DCSS driver add function for a DCSS of type SR

Observation
The time for initial access of the DCSS is a linear function of the DCSS size. The times are recorded as real elapsed times with no user time and almost no system time, because the `add` function issues a CP `diag` instruction to perform the operation. CP has to read the DCSS pages from the spool file located on DASD into memory.

Conclusions
Adding the DCSS of type SR to the Linux address space has significantly larger overhead than adding a DCSS of type SN. This difference reflects the fact that one Linux command makes z/VM read the whole DCSS of type SR from the spool file into memory. The number of pages read per unit time is independent of the DCSS size, therefore the time scales with the size. However, the load time is almost 17 minutes shorter than the time needed to save the DCSS.

Note that the measured time is only for the first Linux guest accessing the DCSS. A second Linux guest accesses the DCSS much faster, because the DCSS pages already reside in memory, and do not need to be fetched from spool space.

Time to copy data into the DCSS

The time to fill a DCSS of type SR is measured for various DCSS sizes.

Figure 6 shows the time to copy data into the DCSS until it is full, for various DCSS sizes.

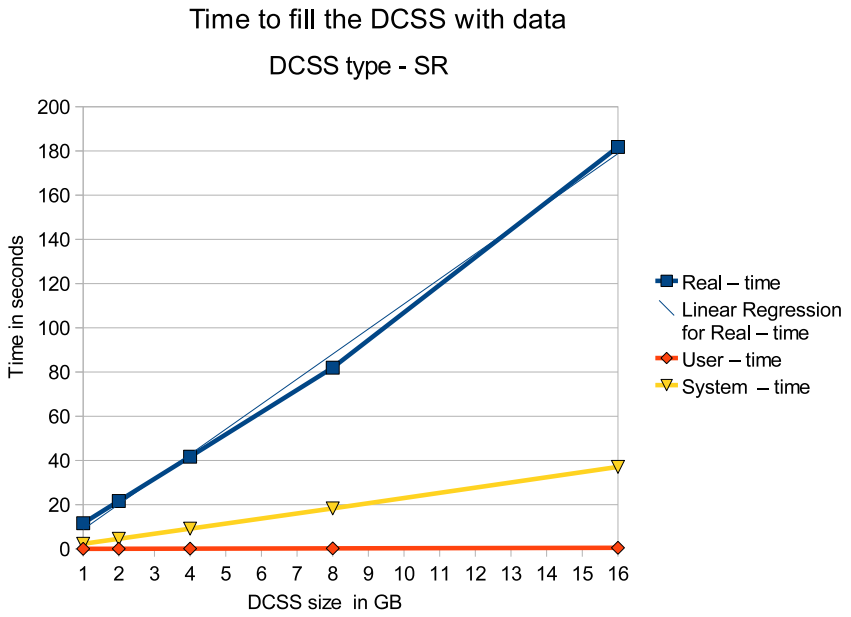

Figure 6. Time to fill a DCSS of type SR

Observations

The time to copy data into the DCSS is a linear function of the DCSS size. There is a large gap between real (elapsed) time, and the system (Linux kernel mode) and user times. This behavior is different from that observed for a DCSS of type SN. The time to complete the copy operation is also longer for the DCSS of type SR.

Conclusions

Compare this graph (Figure 6) with Figure 3. The user and system part is comparable. The real time is much larger for DCSS type SR, indicating that it causes more processing in z/VM than a DCSS of type SN, when updating the contents of the storage.

When considering the times for a DCSS of type SR, take into account that a DCSS of type SR is set up only once, because it is persistent.

Time to complete the Linux save function

The time to save a DCSS of type SR is recorded, for various DCSS sizes.

Figure 7 shows the time for the DCSS `save` function for various DCSS sizes.

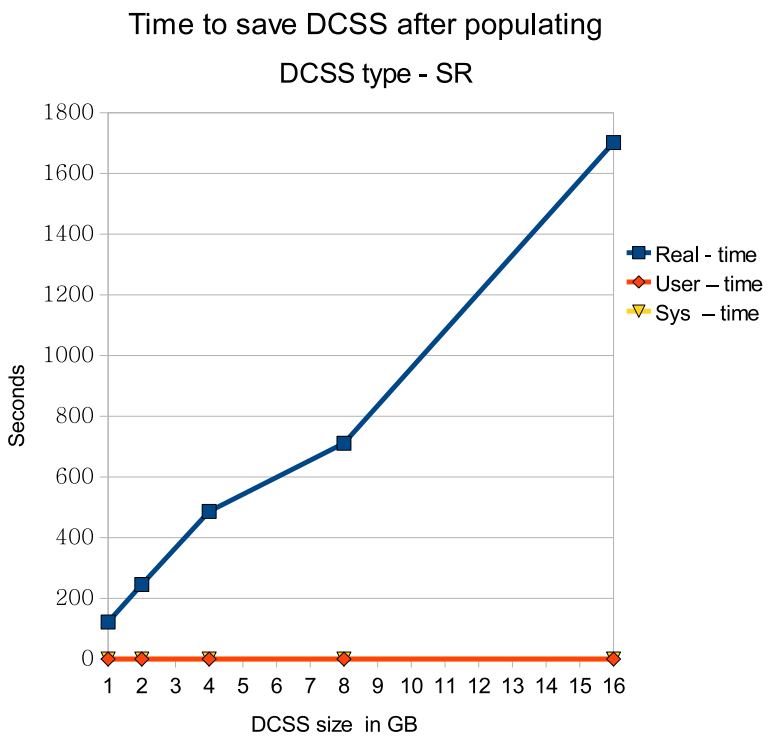

Figure 7. Time to save DCSS of type SR with the DCSS device driver save function

Observations

The save time is almost a linear function of the DCSS size. The `save` function issues a CP `diag` command, which writes the DCSS pages to the spool file. Because the `save` function is performed by CP, almost no Linux system or user time is recorded. The `save` function performed by the Linux DCSS device driver is significantly faster than the save operation performed under CP during the initial DCSS setup.

Conclusion

The `save` function writes the DCSS pages from memory to the spool file. This operation, which updates all pages in the DCSS, takes a little more time than the initial save operation performed by CP.

Time to re-access the DCSS after it was saved

The time to re-access a DCSS of type SR and various sizes, is recorded and analyzed. This was done by removing the DCSS (`echo` *`DCSS name`* `> /sys/devices/dcssblk/remove`) first, and measuring the time needed to add it again (`echo` *`DCSS name`* `> /sys/devices/dcssblk/add`).

Figure 8 shows the time to re-access the DCSS on Linux using the add function of the DCSS device driver for various DCSS sizes.

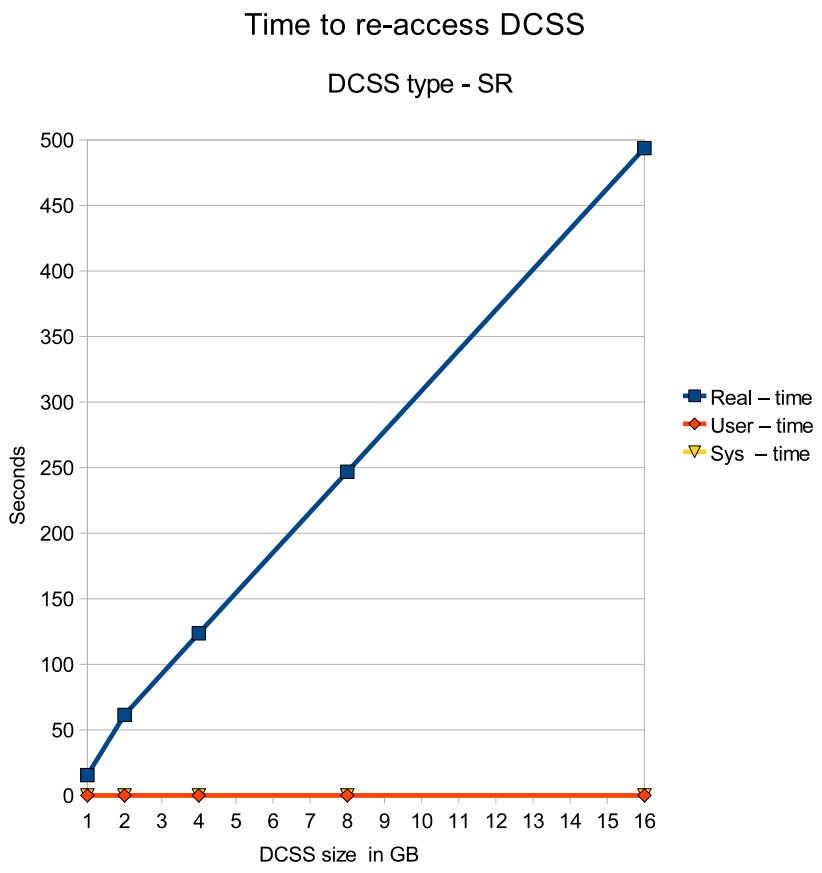

Figure 8. Time to re-access a DCSS of type SR

Observations

The time to re-access the DCSS is a linear function of the DCSS size. The second access of the DCSS by the Linux device driver `add` function is faster than the first one. Again, there is almost no observed Linux kernel time, because the device driver issues a CP `diag` instruction to perform the operation.

Conclusion

The second access of the DCSS is twice as fast as the first access. This implies that z/VM is able to reuse some of the DCSS pages, and does not have to reload them from the spool file.

Summary

Handling a DCSS of type SN is significantly faster than handling a DCSS of type SR, because a DCSS of type SN does not move data to a spool file. In fact, a DCSS of type SN does not save any kind of data, after there are no longer any guests attached to it.

As the size of a DCSS of type SR increases, the time to copy data into it increases linearly, and most the time is spent in z/VM.

The DCSS of type SR requires more steps, because it is saved to the spool file and can be reloaded.

Table 5 summarizes the time needed for the various steps, for an 8 GB and a 16 GB DCSS.

Table 5. Execution times of some tasks for an 8 GB and a 16 GB DCSS

Task	Time to complete task, in minutes	
	8 GB DCSS	16 GB DCSS
CMS save	11	22
First Linux add	8	16
Linux save	12	28
Second Linux add	4	8

A DCSS of type SR has the advantage of being persistent across z/VM restarts. A DCSS of type SR saves time if the construction of the DCSS and the data required to be stored in it is time-consuming. When creating a DCSS of type SR, the most time is used in the save operation, which writes the full DCSS back to the spool. Especially for a DCSS of 16 GB, this write takes a considerable amount of time.

Using a DCSS as swap space

This test case measures the behavior of a DCSS used as swap device. The throughput, CPU cost, and paging operations are analyzed.

The test uses a single guest with a WebSphere Application Server and one database with two WebSphere Application Server workloads. The workloads are DayTrader, which performs read and write operation to a DB2 database, and Bookstore, which performs only read operations. The guest storage is set to 750 MB, a value that causes swapping to occur.

The performance when Linux is swapping to various types of swap device is measured and analyzed. These swap devices are compared:

- A z/VM VDISK of size 1039860 KB
- A DCSS of size 183500 KB
- A DASD device attached to the z/VM guest, swap size 1130392 KB

The scenarios with VDISK and DCSS require additional z/VM memory. Therefore, a fourth test was added:

- Increase the guest memory size by the size of the DCSS, which means from 1024 MB to 2048 MB.

The number of servers was scaled as follows: one, two, four, and eight.

This test case is designed to determine:

- Which type of paging device provides the best performance
- Whether allocating memory for a fast paging device provides an advantage, compared to using this memory to increase the guest size.

Throughput

The throughput is measured when using the DCSS as a swap space.

Figure 9 shows the throughput observed for running both the DayTrader and Bookstore workload on a single guest.

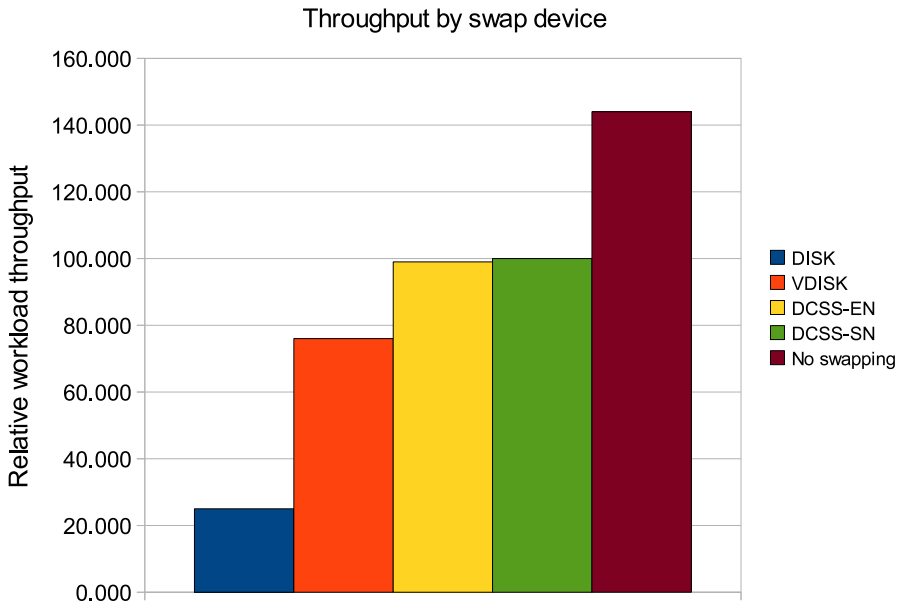

Figure 9. Workload throughput for varying the Linux swap device

Observations

Highest throughput is seen when the memory is used as main memory instead of a memory-based swap device. In fact, the additional memory for this scenario causes Linux not to swap at all. When the memory is used for a swap device, DCSS has the highest throughput, and the physical disk has the lowest throughput. The DCSS type SN has a slightly higher throughput value than the DCSS EW/EN type, and this type was used in the scaling tests.

Conclusion

From the throughput perspective, the use of memory for a fast swapping device should be considered seriously. Using this memory for main memory increases the chance that Linux has no need to swap, which provides the best performance.

CPU cost

The CPU cost is measured when using the DCSS as a swap space.

Figure 10 shows the CPU cost in terms of the total amount of CPU from the LPAR (guest load and z/VM) spent to drive a certain amount of throughput, for the DayTrader workload running on a single guest.

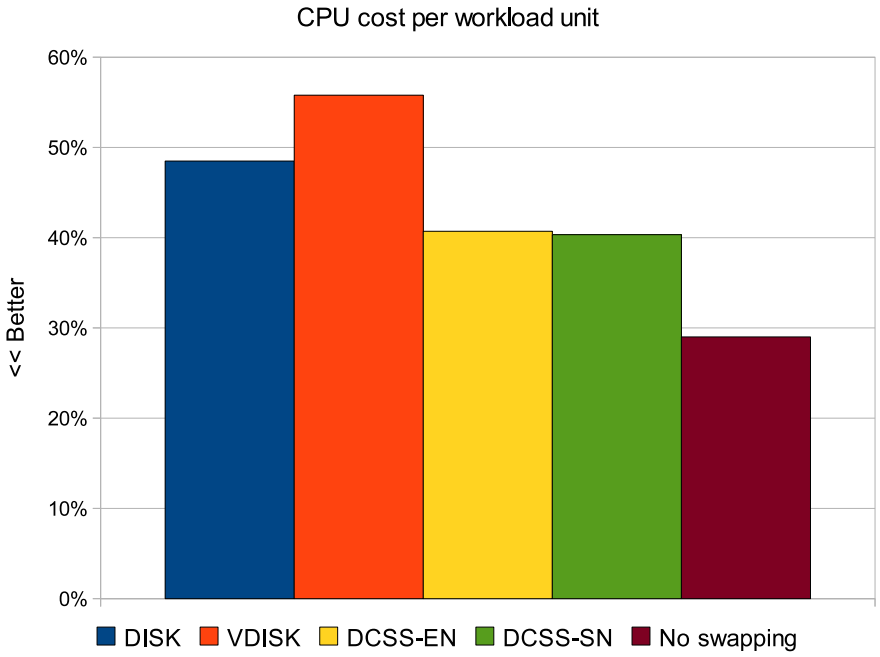

Figure 10. CPU resources spent to drive a certain amount of throughput when varying the Linux swap device

Observations

The smallest amount of CPU to drive the throughput is in the scenario that uses the memory as guest memory, instead of for a swapping device. When using a memory-based swap device, the DCSS has the lowest cost, and the VDISK the highest.

Conclusions

The CPU effort to drive the VDISK seems to be relatively high. The scenario that uses the memory as main memory has the lowest cost, because the effort for the Linux memory management is significantly lower (see Figure 11), especially because there is no need to spend CPU time to identify pages suitable for swapping and move them to the swap device.

The test results show a clear order for the performance of different devices used for swapping. The fastest device is the DCSS, followed by VDISK, and the slowest is the physical disk. However, using the memory to increase the guest size provides the highest throughput.

Page management activity

The paging activity is measured and analyzed when using the DCSS as a swap space.

Figure 11 shows the paging operations when varying the swapping device. Five different paging activities are analyzed for the four types of swap device (disk, DCSS EN, DCSS SN, and VDISK), as well as no swapping at all.

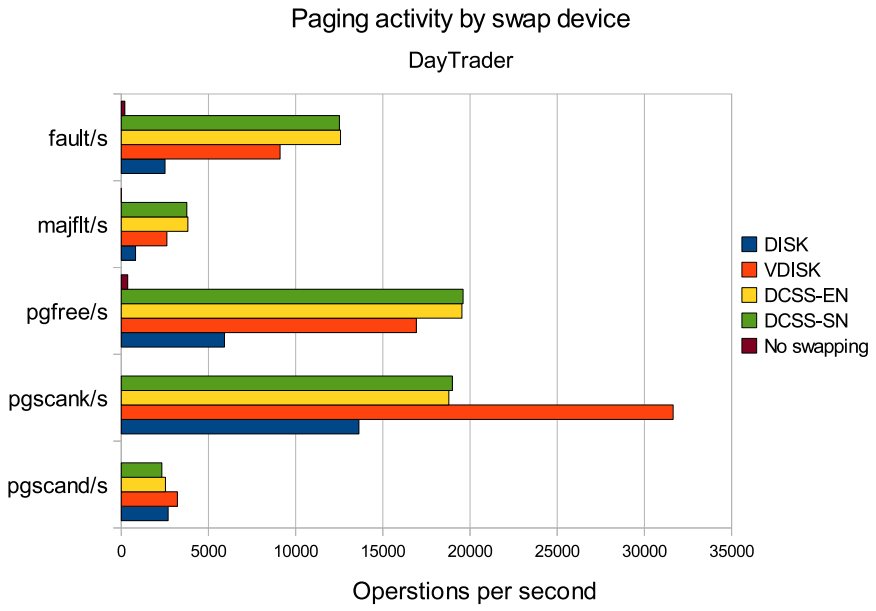

Figure 11. Memory management effort when varying the Linux swap device

Observations

The two most important indicators for the memory activity are major page faults (majflt) and direct scans for pages (pgscand). Major page faults are significant because they must be resolved with disk I/O. Direct scans for pages are significant because, when they take place, an application is waiting for pages. Lower values for these operations mean less effort for the operating system to provide the required memory pages, and fewer interruptions and waits for the application because of missing pages. This means overall higher application throughput at lower CPU cost.

The environment that uses the memory for guest memory, instead of for a fast paging device, has the lowest values. Using a disk or a VDISK behaves comparable in regard to memory management effort. The environment with the DCSS has very high values here.

Conclusions

High values for major page faults and direct page scans indicate waiting processes, which cause lower throughput. In general, this also means a higher effort for the memory management and high CPU cost, which is consistent with the result in Figure 10. It is clear that from the memory management perspective, VDISK or disk behave similar because how the swapping device is implemented is transparent to Linux. One reason for the high cost when using a DCSS is the use of a dedicated driver in Linux.

Scaling the number of servers

This test compares throughput by swap device type as the number of WebSphere servers are scaled. The results are shown for one, two, four, and eight WebSphere servers.

Figure 12 shows the relative throughput when scaling the number of guests: one, two, four, and eight. The data is measured for no swapping, and swapping devices of DCSS of type SN, disk, and VDISK.

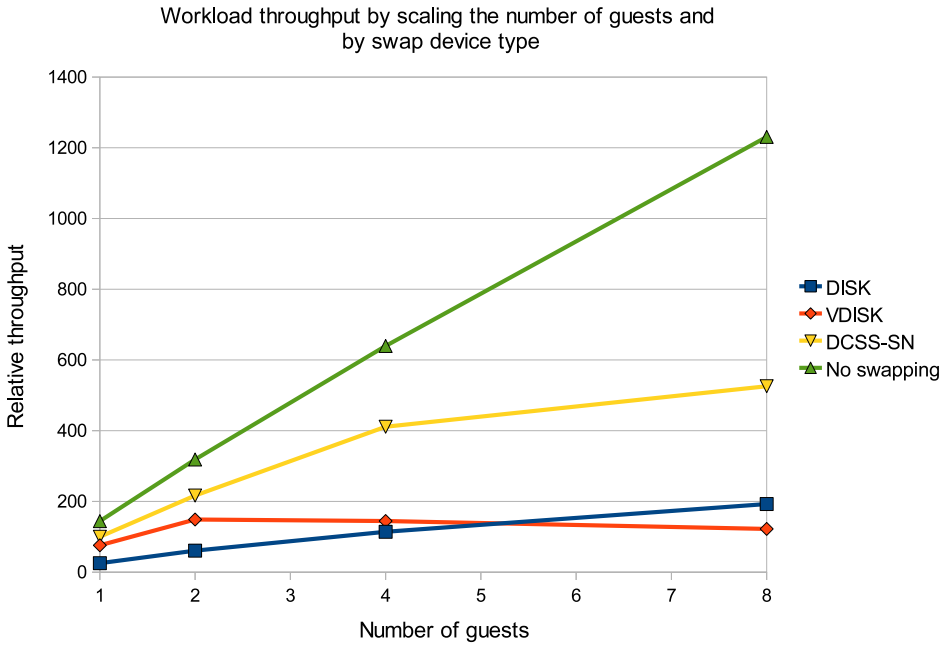

Figure 12. Workload throughput for varying the Linux swap device and scaling the number of guests

Observations

Throughput increases for all swap device types from one to two servers. At four guests, the VDISK swap device does not show a throughput increase. The no swap case has the highest scaling factor, followed by DCSS, and then DISK. While, DCSS throughput did increase fairly linearly from four to eight servers, the increase was less than from one to two and then four. The non-swapping case shows a consistent scaling pattern.

Conclusions

The best swap device was the DCSS of type SN, even though the increase in throughput diminishes with eight servers. Be aware that each guest has its own DCSS. There is no contention for CPU or memory in z/VM for that scenario.

For this environment, using VDISK devices for swapping does not scale. The reason that VDISK has a problem with many guests is related to lock contention.

If swap space is needed in a multiserver environment, then it should be either assigned to DISK or DCSS, but the best result was obtained when the memory is used to increase the guest instead of a memory-based swap device.

Summary

The use of a DCSS used as swapping device is summarized.

VDISK and DCSS are memory-based devices, therefore it is expected that they are faster than a physical DASD device. The DCSS is faster than the VDISK. However, increasing the guest memory size provided the best performance improvement, because it reduced the requirement from Linux to swap memory. The increase of the guest size also reduced the effort used for memory management. The use of the memory for increased guest size is a better trade-off than using this memory for a fast swapping device.

The use of the VDISK or DCSS provides an advantage when used as hot standby swap resource, which is used only for short peak time periods. In this situation, the memory is most of the time not allocated from z/VM or in main memory. The surplus memory of a guest is mostly used, especially when it is doing file I/O.

The recommendation is, in case of permanent swapping activity, increasing the guest size is the better way to improve the performance. In a system that is mostly not swapping, a memory-based swap device to cover load peaks performs better than a disk. When using a memory-based swap device, the DCSS was the better choice for the tested workloads.

Note: All these devices used as swap devices are exclusive and cannot be shared between the guests.

Determining the memory saving using a DCSS

This test compared the performance achieved for five WebSphere Linux guests driven by the DayTrader workload, when the WebSphere executable was in a DCSS or on minidisk.

The z/VM LPAR memory was set to a value to cause z/VM memory pressure. Then, the tests were run either with the WebSphere executable mounted from a DCSS using the execute-in-place option, or from a shared minidisk. Without the DCSS, the system has more memory available, which is used to increase the guest size in 5 MB increments.

Table 6 describes the different test environments.

Table 6. Test environment details

WebSphere executable location	Linux guest memory size
DCSS	768 MB
Shared minidisk	768 MB
Shared minidisk	773 MB
Shared minidisk	778 MB
Shared minidisk	783 MB

Throughput comparison and transactions per IFL

The DayTrader workload is analyzed using the DCSS to hold the WebSphere executable, and without the DCSS but with five different Linux guest sizes. Without the DCSS, the WebSphere executable is located on disk.

For all of these graphs, the word 'Disk' is used to indicate that the WebSphere executable is located on disk, not on a DCSS. The sizes that are listed after the word 'Disk' refer to the Linux guest sizes, not disk sizes.

Figure 13 shows the normalized throughput using the DayTrader workload, with a DCSS holding the WebSphere executable, and without the DCSS but with five different Linux guest sizes.

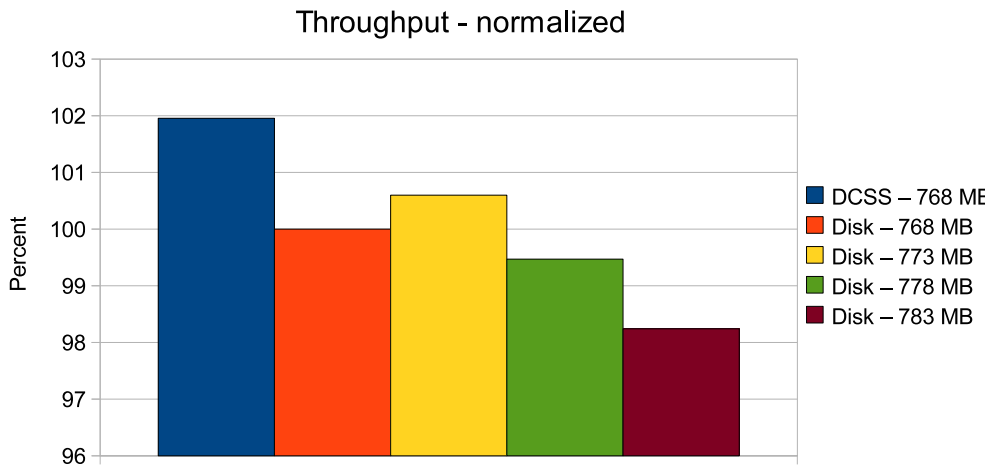

Figure 13. Normalized throughput

Figure 14 shows the number of transactions processed per Integrated Facility for Linux (IFL) using the DayTrader workload, with a DCSS holding the WebSphere executable, and without the DCSS but with five different Linux guest sizes.

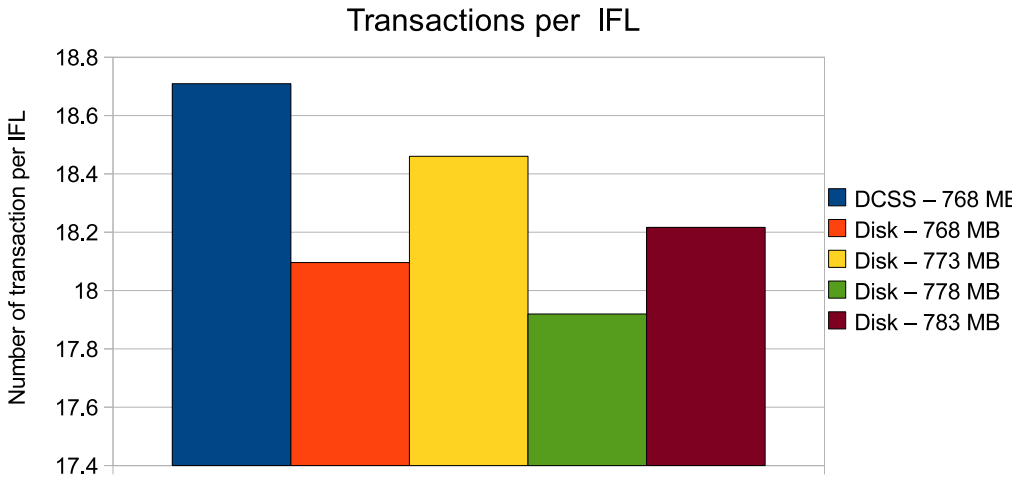

Figure 14. Transactions per IFL

Observations

The throughput of the DCSS environment is higher than any of the 'Disk' environments. When increasing the Linux guest memory size by 5 MB increments, the throughput increases in the first scaling step and decreases with any further memory increase.

Looking at the quantity of transactions driven per IFL shows that the DCSS environment (WebSphere executable located in the DCSS) gets the most out of the used CPUs. When the WebSphere executable is located on disk, the Linux guest with a value of an additional 5 MB has the best efficiency, but this value is still less than the value obtained when using a shared DCSS.

Conclusions

The DCSS uses a certain number of pages that the guests do not provide. This quantity of pages is a constant, depending only on the application used and the workload, but not depending on the number of guests.

Without the DCSS used for the WebSphere executable, there are two competing processes:

- When the size of the guest increases, the workload in Linux has less memory pressure and runs better.
- When the size of the guest increases, the memory pressure in z/VM increases and causes paging, which degrades the performance.

The result from these considerations is that the increase of 5 MB per guest increases the throughput, but an increase of more than 5 MB causes the z/VM effort for swapping to lead to performance degradation. The paging state is shown in Paging statistics.

Paging statistics

Paging statistics are presented for the DayTrader workload when using the DCSS to hold the WebSphere exectuables, and without the DCSS but with four different Linux guest sizes. Without the DCSS, the WebSphere executable is located on disk.

For all of these graphs, the word 'Disk' is used to indicate that the WebSphere executable is located on disk, not on a DCSS. The sizes that are listed after the word 'Disk' refer to the Linux guest sizes, not disk sizes.

Figure 15 show the average number of pages in XSTOR when using the DCSS to hold the WebSphere executable, and four different Linux guest sizes when the WebSphere executable is located on disk.

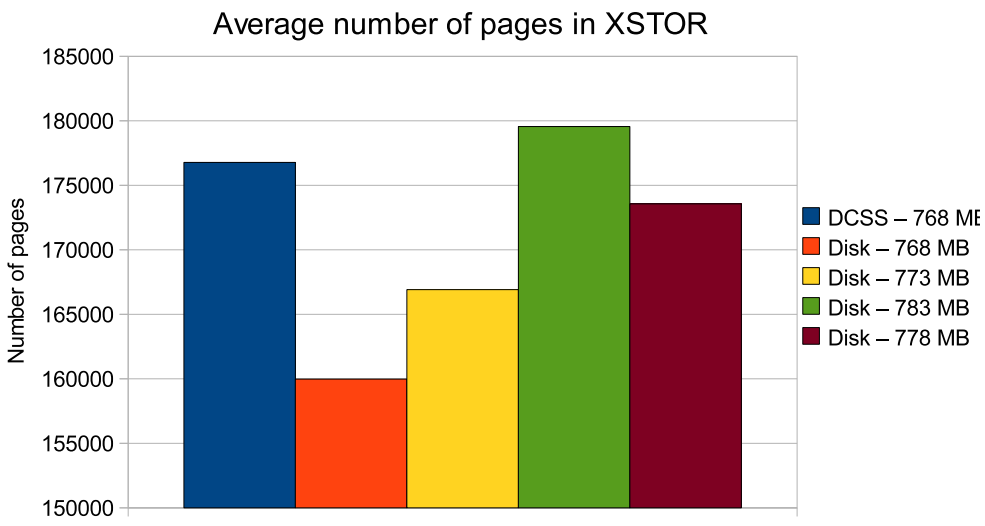

Figure 15. Average pages in XSTOR

Figure 16 shows the average number of pages migrated from main storage to XSTOR using the DCSS to hold the WebSphere executable, and four different Linux guest sizes when the WebSphere executable is located on disk.

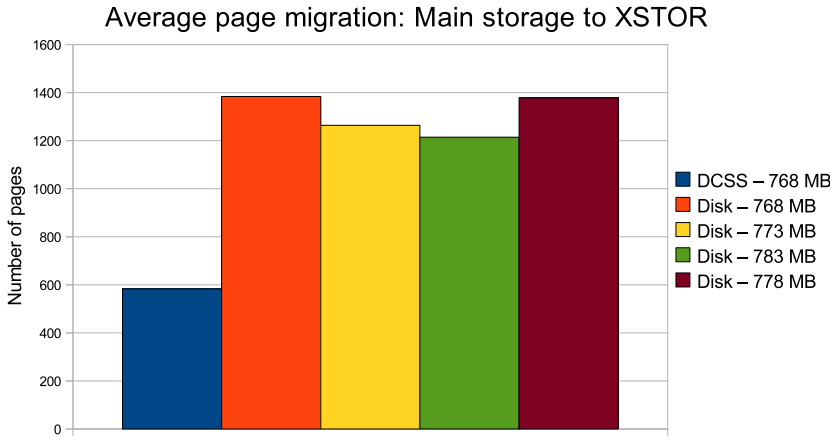

Figure 16. Average page migration: Main storage to XSTOR

Figure 17 shows the average number of pages migrated from XSTOR to main storage when using the DCSS to hold the WebSphere executable, and four different Linux guest sizes when the WebSphere executable is located on disk.

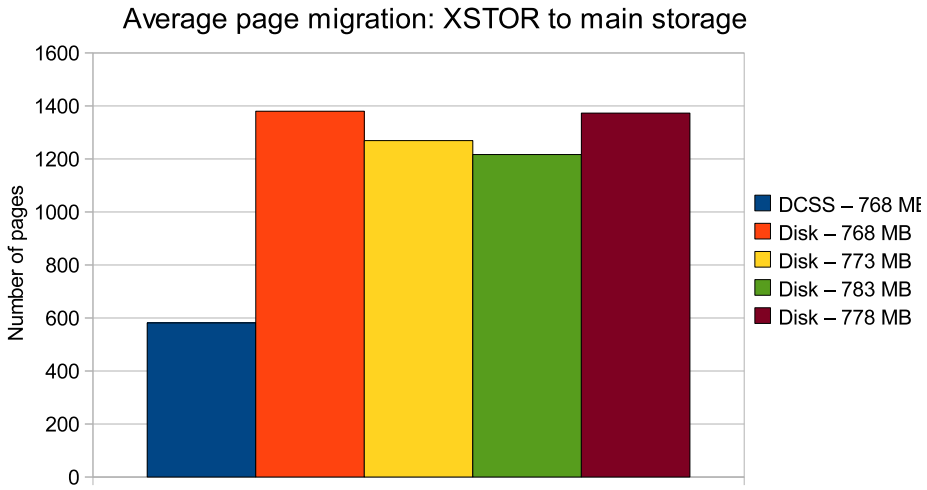

Figure 17. Average page migration: XSTOR to main storage

Observations

When the WebSphere executable is located in the DCSS, there is a large number of pages in XSTOR, but in average a significant reduction in the page migration activities during the test on z/VM (either to or from XSTOR). Without the DCSS, the number of pages in XSTOR is smaller, but this amount increases when increasing the z/VM guest size. In the last scaling step, the number of pages in XSTOR decreases again.

The page migration rates to or from XSTOR are at the highest level when the DCSS is just removed. The migration rate decreases with the increase in guest size, until the last scaling step where it approaches the level where the guest size is the same as with a DCSS.

Conclusions

The paging statistics confirm the assumption that with the increasing guest size the number of pages in the paging space increases. The effort for page migration decreases in the beginning, but then it increases again. Keep in mind that the last scaling step already has reduced memory pressure because of the reduced throughput.

Location of the DCSS pages

Regarding memory savings, the important questions are: how many pages of the 6 GB DCSS are loaded at any time, how many stay resident in main memory, and how many are moved to the paging area.

Figure 18 shows where the pages of a DCSS reside.

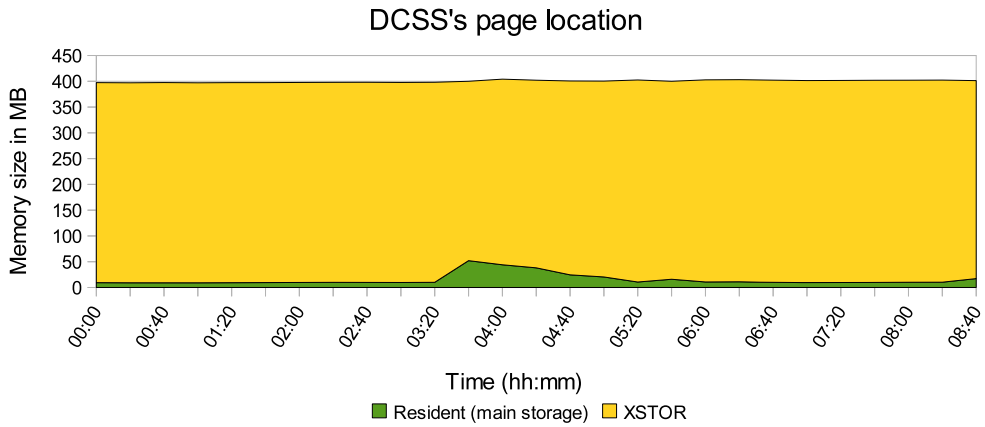

Figure 18. Locations where the pages of the DCSS reside during the test

Observations

From the 6 GB DCSS size, only 400 MB are loaded, and only a small portion stay in main memory. There are no pages on DASD.

Figure 19 shows the amount of page in main memory in detail.

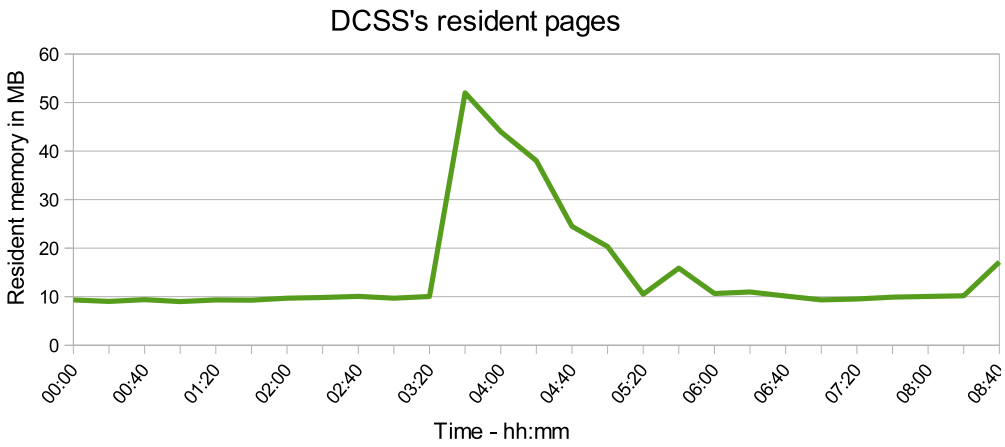

Figure 19. Amount of DCSS pages in main memory (resident) during the test

Figure 20 shows the corresponding transfer rates between the DCSS and XSTOR.

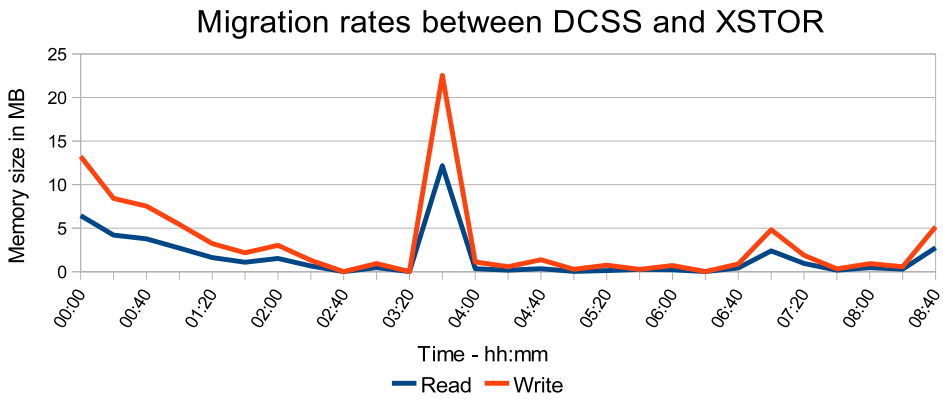

Figure 20. Migration rates in MB per second between DCSS and XSTOR

Observations

Most of the time, 10 MB of the DCSS in main memory is sufficient to drive the test. A peak value is 50 MB of DCSS storage located in main memory. With that setup there is a higher throughput than with any other memory configuration without the DCSS. Also, the migration rates between the DCSS in main memory and XSTOR are very low, there is a peak in resident pages at time 03:40. But even the highest value here, of approximately 25 MB per second is a very moderate value.

Conclusions

Only 400 MB from the 6 GB space of the DCSS is loaded, and it seems that there is some kind of selection of the most needed pages. The number of pages that are needed at runtime and therefore needed in real memory is 10 MB, a low value. Also, the low transfer rates between DCSS and XSTOR indicate that the right pages stay in real memory.

With regards to the memory savings, it is observed that most of the time approximately 10 MB from the DCSS is required. These pages are shared by each guest. To identify the impact of the memory savings using a DCSS, the throughput results are compared with a scenario without a DCSS with the same guest size and with an increased guests size to compensate for the additional memory needed for the WebSphere binaries. Adding 10 MB of memory, as used from the DCSS in the test before, to each guest produces a suboptimal result because of the increase in memory pressure inside z/VM due to the memory being taken from z/VM to give to the guests.

Adding only 5 MB per guest results in the best performance without the DCSS, but this configuration performs still worse than with the DCSS being used to hold the WebSphere executable. The conclusion is that, with only five WebSphere Application Server guests that also include a DB2, it can be recommended to run the WebSphere binaries with the execute-in-place option from the DCSS. The memory saving is at a minimum approximately 10 MB per guest, which would result in a total of 1 GB for 100 guests.

Summary

The DCSS uses a certain number of pages that the guests do not need to provide. Under memory pressure these pages are moved to expanded storage. They are not moved to the paging DASDs because they could be read from spool. Finally, approximately 10 MB stays resident in memory. Even that number is a small value, this number of pages is a constant, depending only on the application used and the workload, but not depending on the number of guests. With regards to the guests, with only five guests a higher throughput and a lower CPU load and memory pressure inside z/VM is observed when the DCSS is used. Comparable values were not be obtained by increasing the guest size. Also, this advantage of the DCSS increases with the number of guests sharing the DCSS.

Sharing read-only data using DCSS

In this test case, five guests are used with a read-only database. The performance of the Swingbench workload when the read-only database is in a DCSS is compared to when the read-only database is on a shared minidisk.

After the database was built, it was copied to a DCSS and the DCSS was mounted in read-only mode.

Table 7 shows the observed results.

Table 7. Comparison of using the DCSS to share read-only data versus database on disk

Database location	Normalized throughput
Shared minidisk, no minidisk cache	100%
Shared minidisk with minidisk cache	119%
DCSS	991%

Turning on minidisk caching results in a 19% improvement compared to the case without minidisk caching. With the database in a DCSS, there is an 890% improvement compared to the minidisk case.

Figure 21 shows the CPU cost for the five guests when the shared database is on a DCSS versus a minidisk with and without minidisk caching.

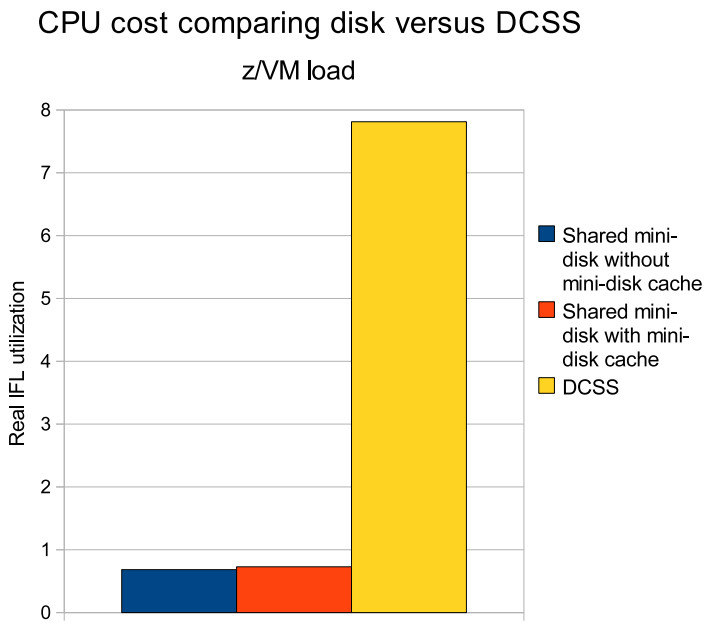

Figure 21. z/VM CPU utilization: Comparing DCSS versus disk for read-only data

The z/VM CPU utilization for the three cases shows that with the database in a DCSS there is significantly more CPU utilization. The cases where the database reside on a shared minidisk has approximately 1/8 the CPU utilization with 1/10 of the throughput.

Figure 22 shows the average of the Linux CPU data for the five guests with the shared database located on the DCSS versus on a minidisk, and with and without minidisk caching.

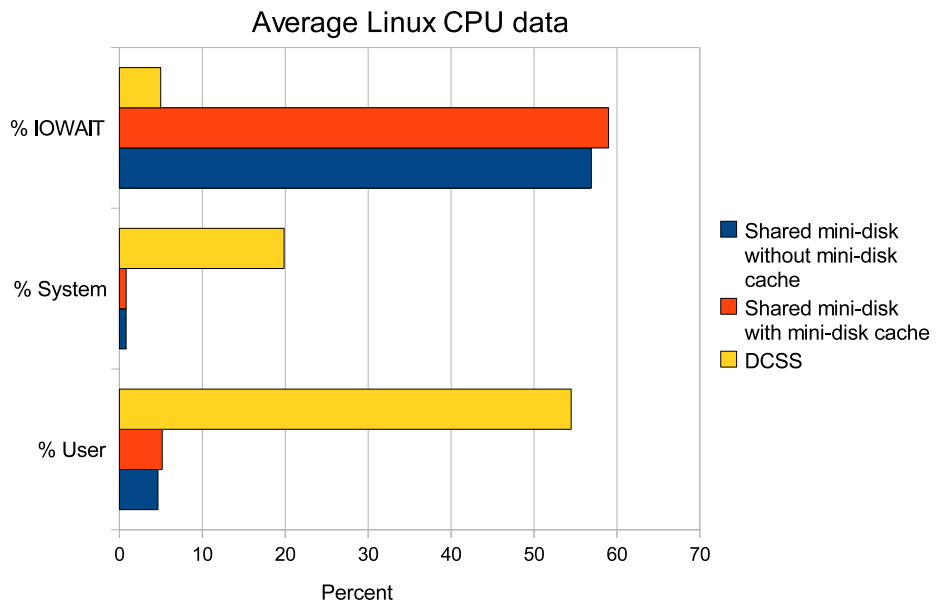

Figure 22. Average of the Linux CPU data

Observation

The use of a DCSS for holding the read-only database shows a major advantage compared to hosting the database on shared minidisks, with almost a factor of 10 improvement in throughput. The database on a shared minidisk is spending half of the time in an IOWAIT state.

Conclusion

The use of a DCSS for holding a shared read-only database has a major advantage compared to shared minidisk, and is highly recommended. Also, minidisk caching has a performance advantage compared to the non-caching case.

Appendix. The CP DEFSEG and SAVESEG commands

Details about the CP DEFSEG and SAVESEG commands, relative to DCSS definition, are provided.

DEFSEG

The DEFSEG command is used to define a skeleton system data file (class S, SDF) for a saved segment. The skeleton file consists of a descriptor page containing all information necessary to preserve the saved segment with the SAVESEG command. A saved segment cannot be accessed until it has been saved.

The DEFSEG command has the following syntax:

DEFSEG *dcssname hexpage1-hexpage2 type options*

where

dcssname
The name of the saved segment to be defined.

hexpage1, hexpage2
A range of pages to be saved.

type
Refers to the page descriptor code of the page range in the saved segment. It indicates the type of virtual machine access permitted to pages in the range.

options
One or more keyword options, such as LOADNSHR and RSTD.

The following types of DCSS segments can be defined (page descriptors):

EW
Exclusive read/write access

EN
Exclusive read/write access, no data saved

ER
Exclusive read-only access

SW
Shared read/write access

SN
Shared read/write access, no data saved

SR
Shared read-only access

SC
CP writable pages, shared read-only access by virtual machine, no data saved. Please note that the SC page descriptor code cannot be used with the LOADNSHR option.

The test systems used in this paper defined DCSS page segments of type SN and SR. When defining a DCSS of type SR, the LOADNSHR option was also used. The LOADNSHR option indicates that any user can load a nonshared copy of the saved segment. No NAMESAVE directory authorization is required. If any member saved segment is defined with LOADNSHR, a nonshared copy of the space itself or any member can be loaded by any user. The LOADNSHR option cannot be used with the RSTD (restricted saved segment) option. The LOADNSHR option also cannot be used with the SC page range descriptor code.

SAVESEG
The SAVESEG command is the final step of a saved segment build process. This command saves the page range areas previously specified by a DEFSEG command. The areas contain the appropriate objects, such as code or data, that were loaded into these defined areas by an installation procedure.

The SAVESEG command has the following syntax:

SAVESEG *dcssname*

where

dcssname
The name of the segment to be saved. This is the file name of the file previously defined with the DEFSEG command.

For more information about DEFSEG and SAVESEG, see

z/VM: CP Commands and Utilities Reference
http://publib.boulder.ibm.com/infocenter/zvm/v5r3/index.jsp?topic=/com.ibm.zvm.v53.hcpa0/hcsf8b22124.htm

Examples

These example DEFSEG commands are used to define the various sizes of DCSS used in this paper.

Note: The DCSS names used in the DEFSEG command must be unique.

For a 1 GB DCSS of type SN:
cp defseg dcss1g 20000-5FFFF sn

For a 1 GB DCSS of type SR:
cp defseg dcss1g 20000-5FFFF sr loadnshr

For a 2 GB DCSS of type SN:
cp defseg dcss2g 20000-9FEFF sn

For a 2 GB DCSS of type SR:
cp defseg dcss2g 20000-9FEFF sr loadnshr

For a 4 GB DCSS of type SN:
cp defseg dcss4g1 20000-9FEFF sn
cp defseg dcss4g2 9FF00-11FDFF sn

For a 4 GB DCSS of type SR:
cp defseg dcss4g1 20000-9FEFF sr loadnshr
cp defseg dcss4g2 9FF00-11FDFF sr loadnshr
For an 8 GB DCSS of type SN:
cp defseg dcss8g1 20000-9FEFF sn
cp defseg dcss8g2 9FF00-11FDFF sn
cp defseg dcss8g3 11FE00-19FCFF sn
cp defseg dcss8g4 19FD00-21FBFF sn

For an 8 GB DCSS of type SR:
cp defseg dcss8g1 20000-9FEFF sr loadnshr
cp defseg dcss8g2 9FF00-11FDFF sr loadnshr
cp defseg dcss8g3 11FE00-19FCFF sr loadnshr
cp defseg dcss8g4 19FD00-21FBFF sr loadnshr

For a 16 GB DCSS of type SN:
cp defseg dcss16g1 20000-9FEFF sn
cp defseg dcss16g2 9FF00-11FDFF sn
cp defseg dcss16g3 11FE00-19FCFF sn
cp defseg dcss16g4 19FD00-21FBFF sn
cp defseg dcss16g5 21FC00-29FAFF sn
cp defseg dcss16g6 29FB00-31F9FF sn
cp defseg dcss16g7 31FA00-39F8FF sn
cp defseg dcss16g8 39F900-41F7FF sn

For a 16 GB DCSS of type SR:
cp defseg dcss16g1 20000-9FEFF sr loadnshr
cp defseg dcss16g2 9FF00-11FDFF sr loadnshr
cp defseg dcss16g3 11FE00-19FCFF sr loadnshr
cp defseg dcss16g4 19FD00-21FBFF sr loadnshr
cp defseg dcss16g5 21FC00-29FAFF sr loadnshr
cp defseg dcss16g6 29FB00-31F9FF sr loadnshr
cp defseg dcss16g7 31FA00-39F8FF sr loadnshr
cp defseg dcss16g8 39F900-41F7FF sr loadnshr

Bibliography

Sharing a WebSphere Application Server V7 installation among many Linux for IBM System z systems
ftp://ftp.software.ibm.com/common/ssi/sa/wh/n/zsw03055usen/ZSW03055USEN.PDF

Or
http://ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101817

How to use Execute-in-Place Technology with Linux on z/VM
http://public.dhe.ibm.com/software/dw/linux390/docu/l26dhe00.pdf

Managing Linux on System z: Using device drivers, features, and commands, SC33-8411
http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/index.jsp?topic=/liaab/concepts/lcon_Managing_Linux_on_System_z.htm

Sharing and Maintaining SLES 11 Linux under z/VM using DCSSs and an NSS
http://www.vm.ibm.com/linux/dcss/ror-s11.pdf

Using Discontiguous Shared Segments and XIP2 Filesystems With Oracle Database 10*g* on Linux for IBM System z http://www.redbooks.ibm.com/redbooks/pdfs/sg247285.pdf

z/VM: CP Commands and Utilities Reference, SC24-6081
http://publib.boulder.ibm.com/infocenter/zvm/v5r3/index.jsp?topic=/com.ibm.zvm.v53.hcpa0/hcsf8b22124.htm or http://publibz.boulder.ibm.com/epubs/pdf/hcse4b31.pdf

z/VM Saved Segments Planning and Administration, SC24-6116-02
http://publib.boulder.ibm.com/infocenter/zvm/v6r1/index.jsp or
http://publibz.boulder.ibm.com/epubs/pdf/hcsg4b10.pdf

Glossary

Architected segment
A 1 MB portion of real storage defined by ESA/390, ESA/XC, and z/Architecture®.

Discontiguous saved segment (DCSS)
A discontiguous saved segment (DCSS) is a saved segment that can be embedded above the virtual machine's defined storage size. A DCSS occupies one or more architected segments, and is accessed by name. Although individual address ranges are specified on page boundaries anywhere within an architected segment, a DCSS begins and ends on a 1 MB boundary.

eXecute-In-Place (XIP)
The Linux XIP technology is used to treat code in a memory-backed file system as if it were a part of the virtual memory space. Instead of being loaded into Linux guest memory, executables residing in a DCSS are run directly from the DCSS memory.

Member saved segment
A member saved segment is a special type of DCSS that belongs to up to 64 segment spaces. A member saved segment begins and ends on a page boundary and is accessed either by its own 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. However, the virtual machine should load the other members before trying to use them.

Named Saved System (NSS)
A z/VM technology used to save a bootable operating system snapshot to the z/VM spool. This system snapshot can be shared and started by many z/VM guests, all of which can boot it like an operating system disk and run the system therein.

Usually, an NSS is created at a very early stage in the initialization of the operating system. This enables the operating system in the NSS to detect hardware and other features at the time it is started in each guest.

An NSS must be defined specifically for the operating system image that it will contain. This is because different operating systems have different memory locations which either must be kept exclusive-writable or that can be shared. z/VM keeps track of these mappings, ensuring that guests get their own copy of pages that are writable while giving shared access to read-only pages. In addition, each operating system that creates an NSS must do so with an awareness of what areas to be saved must be read-only and sharable or read/write and exclusive.

While both a DCSS and an NSS are saved as spool files and use the DEFSEG and SAVESEG command, they have different functions and an NSS must have operating system support to function correctly

Saved segment
An area of virtual storage that is assigned a name and saved. Segment spaces, member saved segments, and discontiguous saved segments are defined by CP and saved in system data files. Logical saved segments are defined by CMS. A saved segment can be dynamically attached to and detached from a virtual machine and can be shared by many virtual machines. A z/VM saved segment is a range of pages of virtual storage consisting of a number of memory segments that can hold data or reentrant code (programs).

Segment
A 1 MB portion of real storage as defined by the ESA/390 architecture.

Segment space
A segment space is a special type of DCSS that is composed of up to 64 member saved segments referred to by a single name. A segment space occupies 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 1 MB boundary. A user with access to a segment space has access to all its members.

IBM Systems and Technology Group
Route 100
Somers, New York 10589
U.S.A.
Produced in the United States of America,
03/2011

ZSW03186-USEN-00