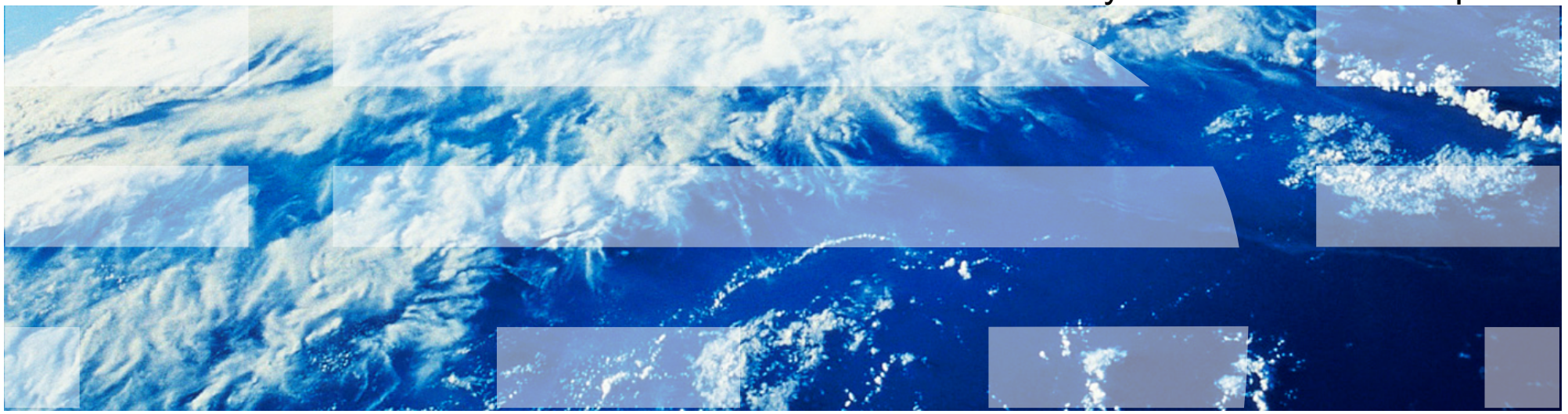
2011-10-25

Linux on System z

Optimizing Resource Utilization for Linux under z/VM - Part1

Dr. Juergen Doelle
IBM Germany Research & Development



visit us at http://www.ibm.com/developerworks/linux/linux390/perf/index.html

Trademarks

Agenda

- Introduction
- z/VM Resource Manager (SRM)
- Recommendations for Memory Overcommitment on z/VM
- Large DCSS
- Summary

Introduction

- **Running virtualized guests provides significant advantages in regard of**
  - Systems management
  - Resource utilization
- **Running many systems with shared resources introduces a new dynamic into system compared to one system on one box**
- **Important questions**
  - Has the right system the right resources at the right moment?
  - Can I bring more guests online or increase guest sizes?
  - When does overcommitment become counterproductive?
- **When are the service level goals not longer met?**
  - Which load situation is acceptable?
- **Objectives of that presentation**
  - What can be done to optimize resource consumption, what are the options
  - Impact on WebSphere Application Sever environments and databases
  - Focus is on memory utilization

Agenda

- Introduction
- **z/VM Resource Manager (SRM)**
- Recommendations for Memory Overcommitment on z/VM
- Large DCSS
- Summary

SRM Considerations for Linux guests

- **The responsibility of the SRM is:**
  - to ensure that guests have the required resources available when scheduled
  - Guests are divided into queues Q1 – Q3 according to their interactivity
    - CMS guests are typically Q1 guests
    - Linux guests are typically Q3 guests
  - When a guest should be scheduled and SRM comes to the decision that the intended amount of resources can not be granted:
    ==> the guest is placed on the eligible list and waits
- **SRM settings might be overruled by enabling QUICKDISPATCH for the guests**
  - But this should be used carefully, e.g. for business critical guests
  - Do not use QUICKDISPATCH for the majority of the guests, this limits z/VM's possibilities to dynamically manage the resource usage
- **Consider that Linux guests are running mostly in Q3!**

SRM Considerations for Linux guests

- **Default SRM settings (`q srm`)**
  - `LDUBUF : Q1=100% Q2=75% Q3=60%`
  - `STORBUF: Q1=125% Q2=105% Q3=95%`
- **LDUBUF – to partition the system's paging resources**
  - 60% for Q3 means:
    All Q3 guests together must use at maximum 60% of the paging resources
    if already used → eligible list
  - Recommendation:
    `SET SRM LDUBUF 100 100 100`
    to allow all Q3 guests to allocate the whole paging space
- **STORBUF – to partition host storage (central storage)**
  - 95% for Q3 means:
    All Q3 guests together must use only 95% of the system storage
    → This prevents memory overcommitment when running Linux guests
  - Recommendation:
    `SET SRM storbuf 300 250 200`
    to allow all Q3 guests to allocate twice the amount of real storage
  - Depending on the level of overcommitment and amount of active/inactive guests, it might be necessary to go even higher, e.g. `SET SRM storbuf 300 300 300`
  - Ensure to have sufficient paging space!

Agenda

- Introduction
- z/VM Resource Manager (SRM)
- **Recommendations for Memory Overcommitment on z/VM**
- Large DCSS
- Summary

Memory overcommitment

- **Memory overcommitment is often mentioned as a major benefit of virtualized environments**

  Memory overcommitment is the ability to use more virtual memory as physically available memory. It is based on the assumption that not all guests use their memory at the same time and/or some guests are over-sized in their memory setup.

- **Recommendations / limits are missing**
  Very different definitions exist for the level of memory overcommitment
  - they are independent of the used middle ware
  - "Common" System z ratio is 3:1 virtual (guest memory) to physical memory
    E.g. run guests defined with a total of 3GB on 1 GB real memory
  - Performance can be heavily degraded when the memory overcommitment level is to high

- **Goal: Proved memory – overcommitment recommendations**
- **Identify "rules"**
  - Determine the minimum amount of physical memory required to run with an acceptable performance
  - Identify a dependency on the used middle ware / applications

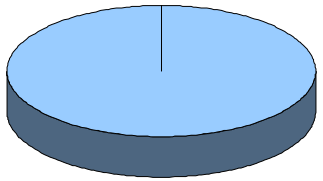
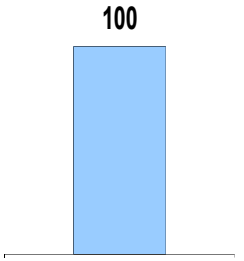
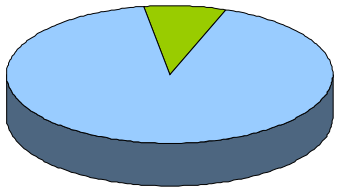
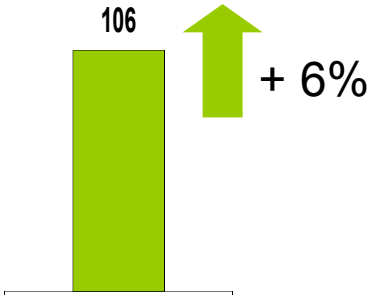
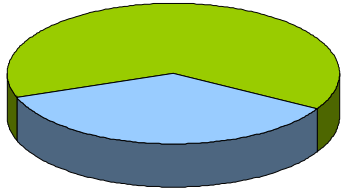
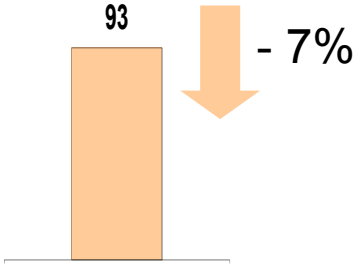
Tested Scenario

- **Test environment**
  - Running a mix of server types as Linux guests on z/VM:
    LPAR with 28GB central storage + 2 GB expanded storage

Guest workload	Guest Memory
WebSphere Application Server	13.5 GB (Java heaps 8GB)
Database DB2	12.0 GB (memory pools about 2 GB)
Tivoli Directory Server (ITDS)	1.5 GB
Idling guest	1.0 GB

- **Test scenarios**
  - Leave the guest size fix
  - Decrease the LPAR size in predefined steps to scale the level on memory overcommitment
  - Measure the execution time of a predefined workload (TPM)

Test Results

Memory – less is better		Performance – more is better
 **100%**	**BASE settings = 100%** • Sum of guest size definition • Base performance	
8% saved 	**OPTIMAL settings** + Reduce memory by 8% + Improved performance by 6%	
64% saved 	**CHEAPEST settings** + Reduce memory by 64% – Decreased performance by 7%	

Results & Recommendation

- **Virtual memory = Physical memory**

  → Does not necessarily provide the best performance (at least not for large LPARs, e.g. 28GB)

- **Optimal memory setting: System shows no z/VM paging activity!**
  - See PerfKit Panels:
    FCX113 User Paging Activity and Storage Utilization (UPAGE)
    FCX254 Available List Management, by Time (AVAILLOG)

- **Estimate minimum memory requirements:**
  - WebSphere Application Server: Sum of all active Java heaps (8GB)
  - DB2: Sum of MAX_PARTITION_MEM (2GB), as reported from:
    "`SELECT * FROM TABLE (SYSPROC.ADMIN_GET_DBP_MEM_USAGE()) AS T`"
    → Value of PEAK_PARTITION_MEM might be used, when highest allocation is captured
  - Linux Page Cache: Is ignored in our case
    Estimation: Sum of read/write throughput, e.g. 20 MB/sec read and 10 MB/sec write throughput require 30 MB/sec pages, is ignored in our case in regard to the 10GB contributed from each of WebSphere and DB2
  - Idling guests: Can be ignored (no kind of server started!)
  - ITDS: Is ignored because of the very small memory footprint
  - Results in 10GB, add 10% Buffer → Minimum System size with good performance was 11GB

Results & Recommendation cont.

- **System runs well in a range from 28 GB to 11 GB**
- **In this environment the sum of all active Java heaps (-Xmx) and allocated DB2 memory pools define the minimum memory size**
  → This is really the lower limit, do not cross!
- **Be aware of the dynamic of a virtualized environment**

  New guests are easily created, Java heaps and database pools might be increased without notifying the System z administrator

- **Monitor paging activity of your system:**
  - FCX113 User Paging Activity and Storage Utilization
  - FCX254 Available List Management → indicates the available memory for new guests
  - FCX135 Average User Wait State Data (→ %PGW)

- **Other workload types/server types might follow similar considerations**

- **More details in "Tivoli Provisioning Manager Version 7.1.1.1: Sizing and Capacity Planning"**
  - http://public.dhe.ibm.com/software/dw/linux390/perf/ZSW03168USEN.PDF
    Chapter 9. Memory overcommitment

Agenda

- Introduction
- z/VM Resource Manager (SRM)
- Recommendations for Memory Overcommitment on z/VM
- **Large DCSS**
- Summary

DCSS

- **Discontiguous Saved Segments (DCSS) is a z/VM technology used to share memory between a number of guests**
- **The DCSS itself has been available in z/VM for a while.**
  The new feature: The size of a DCSS can now exceed the prior limitation of 2 GB
  - Requires SLES11 SP1 or RHEL 6 Linux distribution
- **A DCSS is memory based,**
  - provides very fast access
  - requires real memory
- **Allows sharing of memory pages between guests**
  - reduces the memory requirements for the guest, especially when sharing data (read only)
  - The memory is z/VM managed
- **Supports Execute-in-place (XIP), programs code is executed from the DCSS and is not loaded into the Linux memory (page cache),**
  - Saves Linux memory
- **Can be made persistent, saved to Spool area**
  - Requires sufficient Spool space, disk I/O bandwidth of the Spool DASDs can become an important performance aspect

DCSS usage

Where can a DCSS be used:

- **As fast Swap device**
- **For sharing read only data**
- **For sharing code (e.g. program executables/libraries)**

- **The large DCSS allows the installation of a full middleware stack in the DCSS (WebSphere, DB2, etc)**
  - The DCSS becomes a consistent unit of one software level, avoids inconsistencies between DCSS and local disks

That's the agenda for the next slides, but before, we start with some basics:

- **Types of DCSS**
- **How to create a DCSS**
- **Access and load times**

Types of DCSS

The type definition consists of two characters

- **The first character is either E for exclusive access or S for shared access**
  - Sharing of storage between virtual machines is based on 1 MB segments, so all 256 pages in any one segment must be either exclusive or shared
- **The second character of the page descriptor code defines the storage protection characteristics of the page. Mostly used in our case**
  - N Stands for read/write, no data saved
  - R read-only and saved
  - W read/write access and saved

	DCSS Type	
Consideration	**SN: Shared R/W**	**SR: Shared R/O**
File system loaded into DCSS is written to z/VM spool	No	Yes
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
Spool processing for DCSS can delay other spool activity	No	Yes

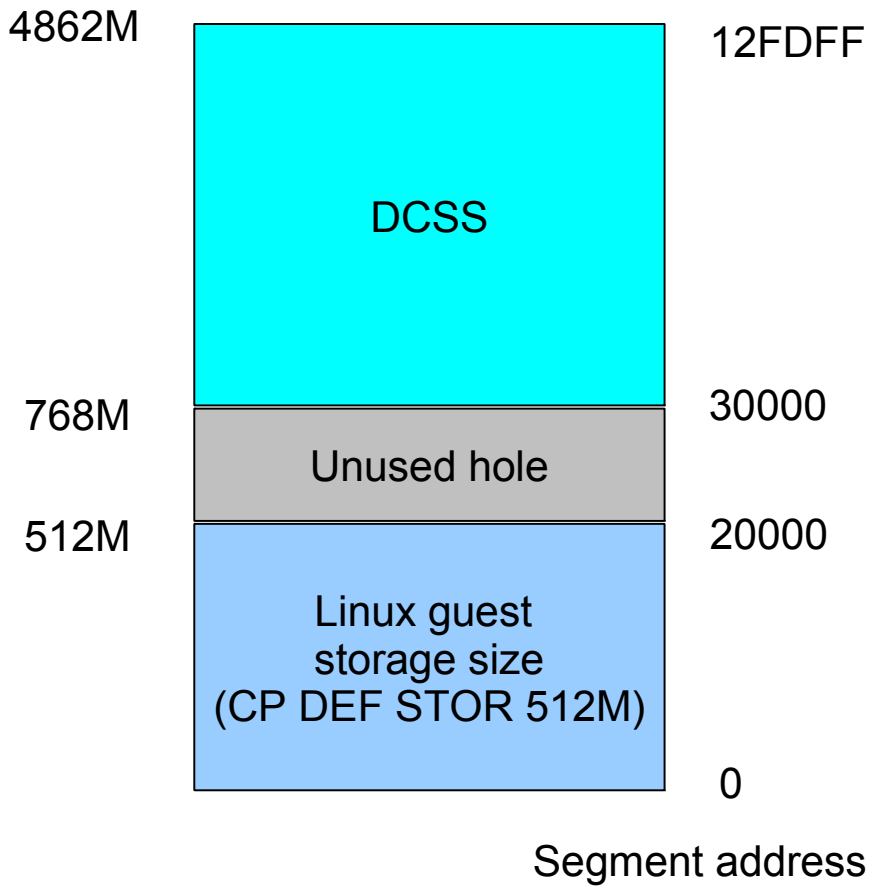
- **Examples:**
  - DCSS for swapping: SN or EN
  - DCSS for initial population: SW or EW
  - DCSS for shared code or data: SR

Defining a DCSS

- **From a CMS user with authentication Class E:**
  - Define a DCSS of type SR and size 4 GB in 2047 MB chunks (units pages)

```
cp defseg dcss4g1 30000-AFEFF  sr
cp defseg dcss4g2 AFF00-12FDFF sr
```

  - The CMS guest which wants to save the new DCSS needs an appropriate size to include the full DCSS size!
  - The spool area must be able to hold three versions of one DCSS (state S, A, P)



- **To edit the DCSS from a linux guest**
  - Add the user directory entry

```
NAMESAVE DCSS4G1 DCSS4G2
```

- **To access the DCSS from a linux guest**
  - Define the guest with a certain size, e.g. 512 MB

```
def stor 512M
```

  - Ensure that Linux memory and DCSS address range do not overlap

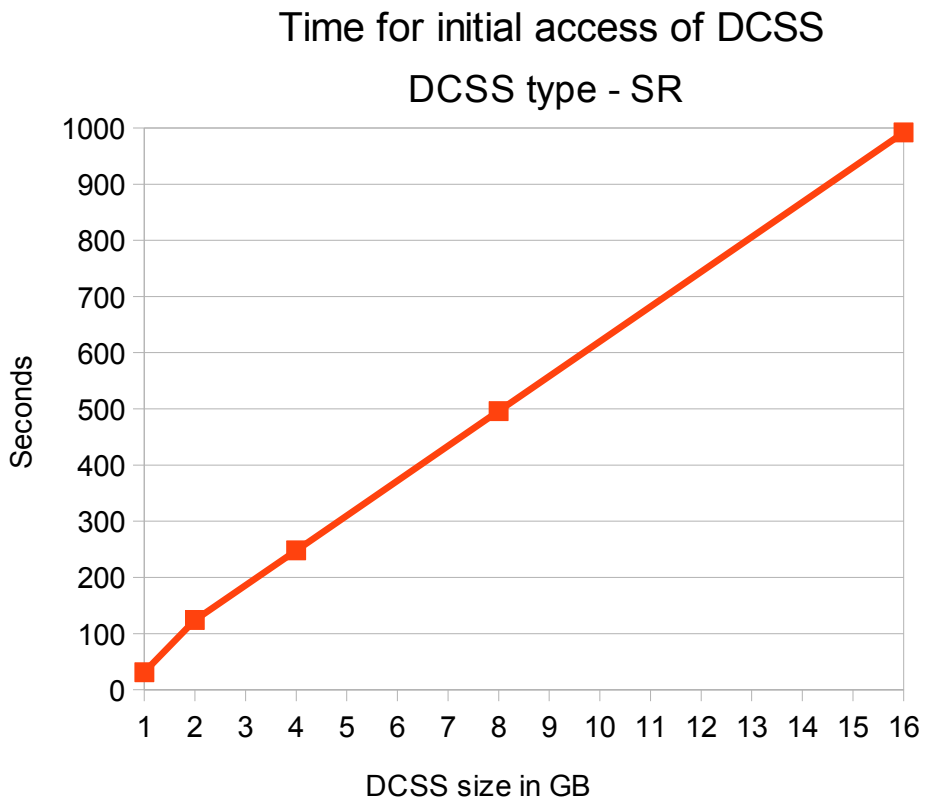
Populating a DCSS

In Linux:

- **load the DCSS block device driver:** `modprobe dcssblk`
- **get the major device number for the dcssblk device:** `cat /proc/devices`
- **create a device node for the DCSS device using the major from above and a free minor**
  `mknod /dev/dcssblk0 b 252 0`
- **Enable the DCSS via** `echo DCSS4G1:DCSS4G2 > /sys/devices/dcssblk/add`
  creates a subdirectory with the name of the DCSS within the /sys/devices/dcssblk directory
- **Make the DCSS writable by setting the shared value to exclusive use (Type SR is mounted ro per default)**
  `echo 0 > /sys/devices/dcssblk/DCSS4G1/shared`
- **Make an ext2 file system on the DCSS:** `mke2fs -b 4096 /dev/dcssblk0`
- **Mount the DCSS:** `mount -t ext2 -o xip /dev/dcssblk0 /mnt`
- **Copy data to the DCSS**
- **Unmount the DCSS** `umount /mnt`
  no administrative operations possible on a mounted DCSS
- **Save the DCSS. The DCSS is so far changed only in memory**
  Save to disk (spool): `echo 1 > /sys/devices/dcssblk/DCSS4G1/save`
- **Remove the DCSS to cause the newly saved DCSS to become active in the spool and purge the old one:**
  `echo "DCSS4G1" > /sys/devices/dcssblk/remove`
- **Enable the DCSS via** `echo DCSS4G1:DCSS4G2 > /sys/devices/dcssblk/add`

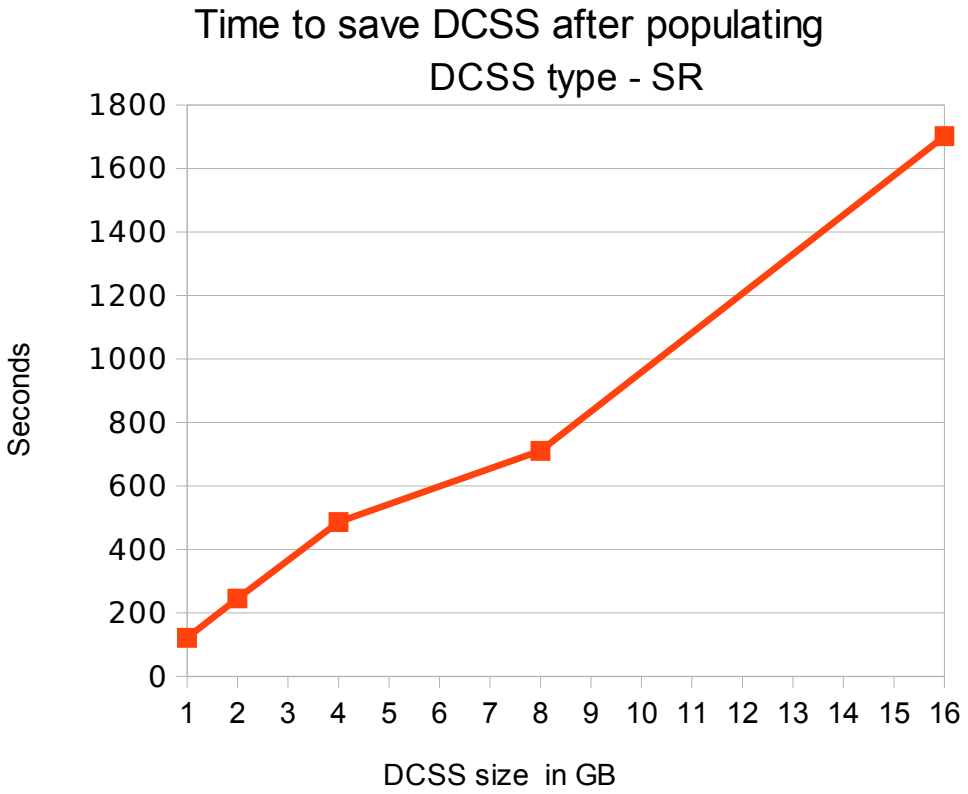
- **Find more information in the Linux device driver book:**
  http://www.ibm.com/developerworks/linux/linux390/documentation_dev.html

Loading a DCSS



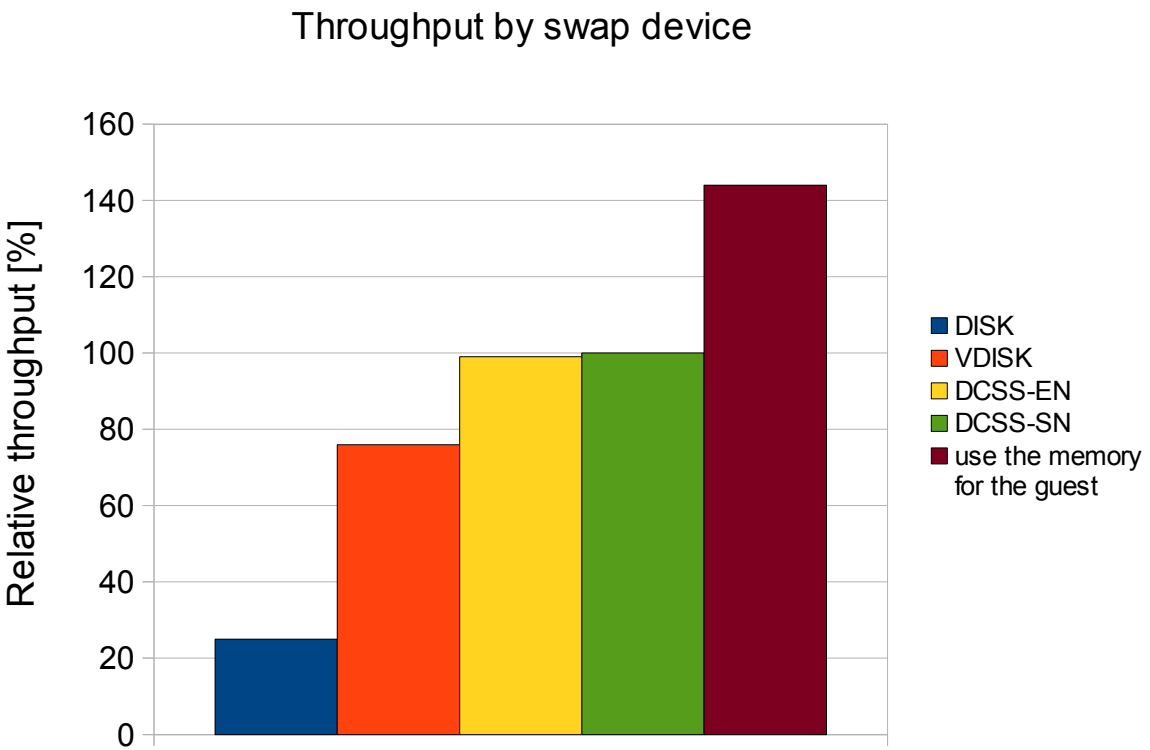
- `echo DCSS4G1:DCSS4G2 > /sys/devices/dcssblk/add`
- **This may take a while for the first time, be patient!**
  Spool file system I/O bandwidth is important here
- **Adding the DCSS causes only a minor amount of pages to be allocated (100-200)**
- **Hits only the first guest, which adds the DCSS. Succeeding guests get it immediately**
- **Type SN required 5 – 35 seconds**

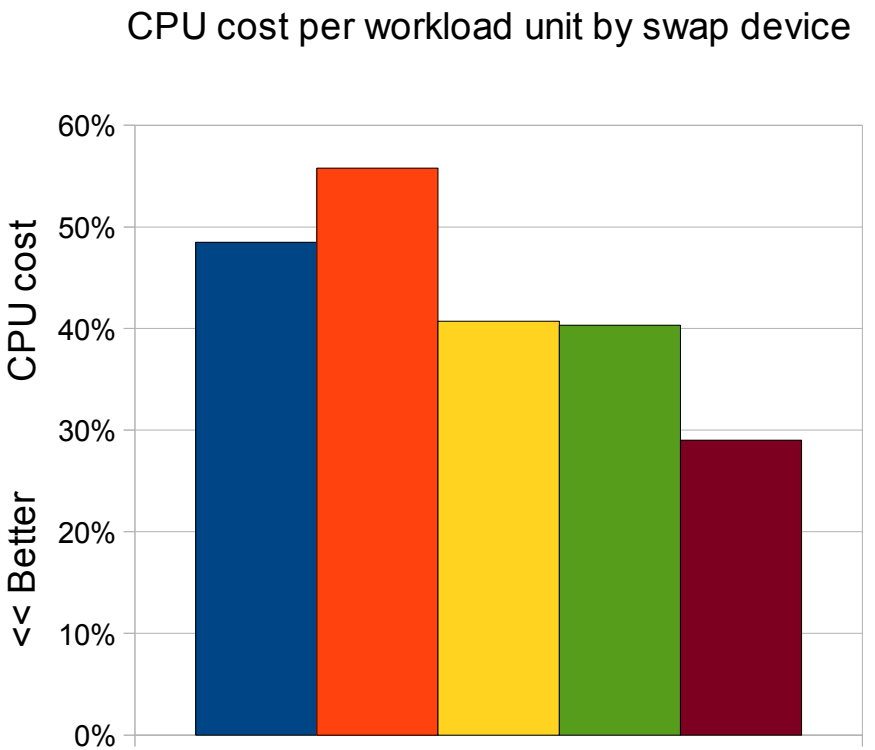
Saving a DCSS



- `echo 1 > /sys/devices/dcssblk/DCSS4G1/save` **This may take a while, be patient!**
- **Saving a DCSS causes**
  - the creation of a new segment (type S, changes to A during save, while the old one changes from A to P)
    - The other guests continue to work with the old version
    - The segment with the type P will be removed once the last guest has released it
  - the allocation of all pages of the DCSS in memory (!)
- **Removing the DCSS from the virtual memory of the guest releases the allocated memory**

DCSS as Swap Device - Guest and z/VM was under memory pressure





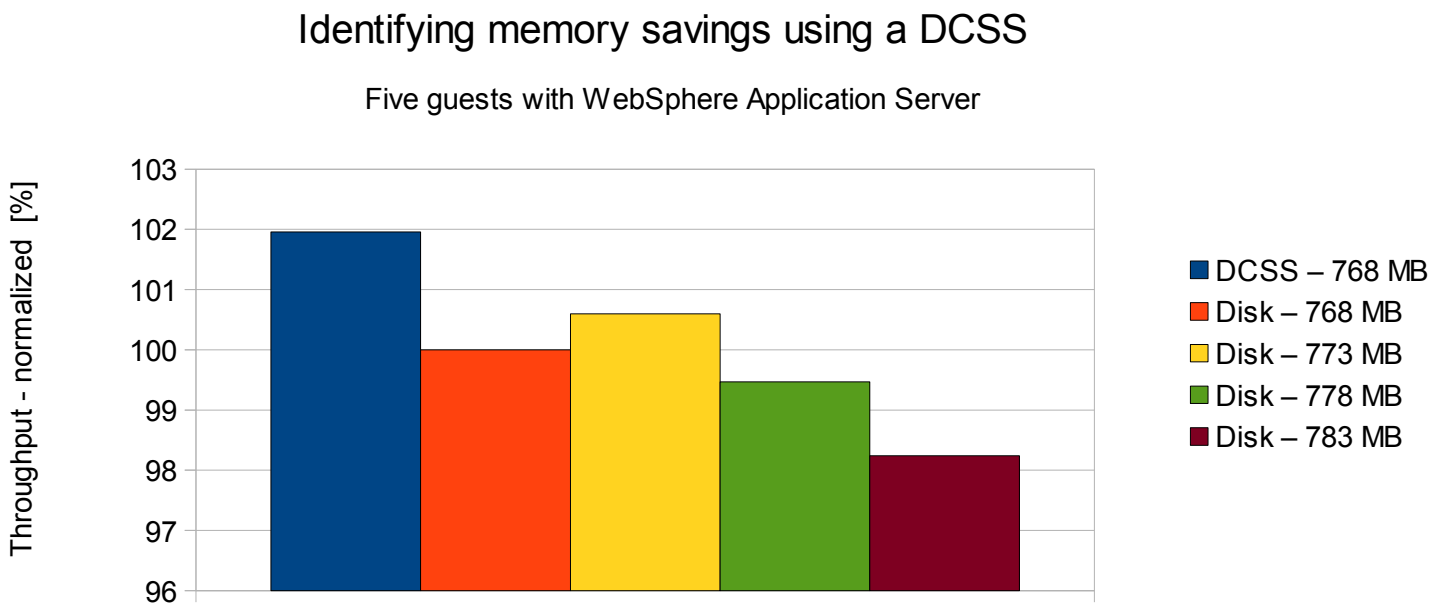
- **In case a memory based swap device is used, the DCSS provides best throughput and at lowest CPU cost (guest stays in SIE instruction)**
- **The best throughput at lowest CPU cost per workload unit was reached with using the storage as guest memory**
  - This setup prevented the need for the guest to swap
- **Recommendation:**
  - When the guest is permanently under memory pressure it is probably better to give it more memory
  - For short peaks in memory pressure a memory based swap device might be appropriate as fast swapping device
    - Should be z/VM based, that the memory pages are allocated only when really in use
    - The peak should not appear on all guest at the same time
    - Then a DCSS is a good solution: it is fast, with small overhead (smaller than with a disk!)

Sharing Read Only Data

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

- **Sharing a read only database with five guests**
- **Provides a throughput improvement of factor 10x compared to a shared disk!**
- **Recommendation:**
  - A DCSS is the first choice for sharing read only data, not updated too often

Can we save memory in WebSphere Application Server Environments?

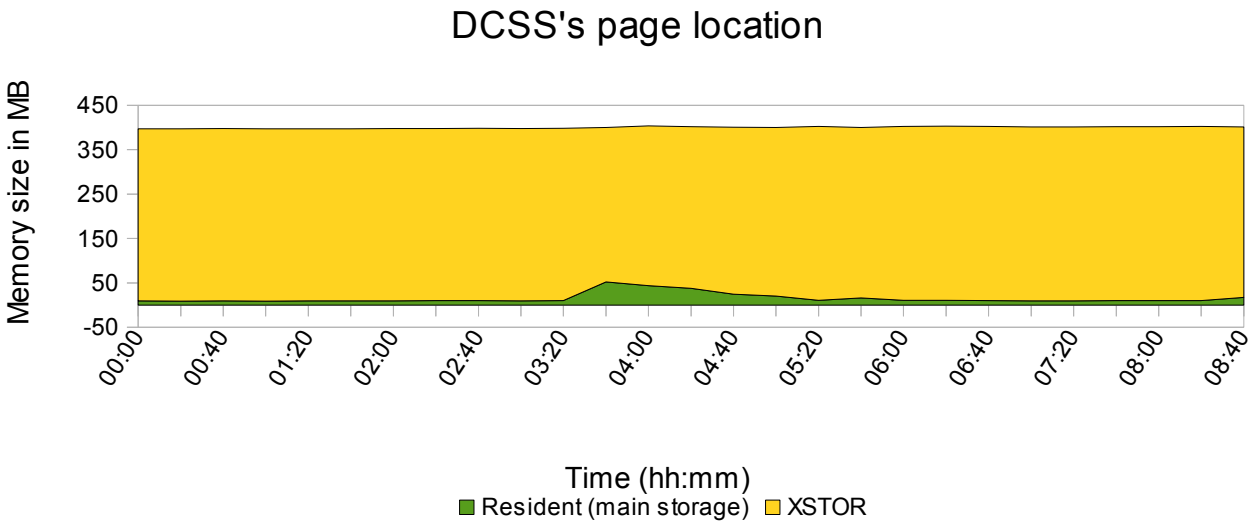


- **Methodology:**
  - Installing WebSphere and DB2 in a 6GB DCSS, mounting the DCSS with -o xip
  - Installing WebSphere and DB2 on a shared disk without DCSS (provides in theory 6GB additional memory)
  - z/VM is under memory pressure

Idea: Scale the guest size without DCSS until throughput from the setup with DCSS case is reached

- **Results:**
  - Best result with the WebSphere on disk is provided with 5 MB more per guest, but still below the DCSS result
  - Increasing the guests soon causes z/VM paging to DASD and accordingly a performance degradation ==> with 50 MB more guest storage the memory pressure is already worse than with the DCSS
  - With only 5 guests we see already a remarkable advantage for the DCSS
    - Saving must be above 25 MB totally for five guest

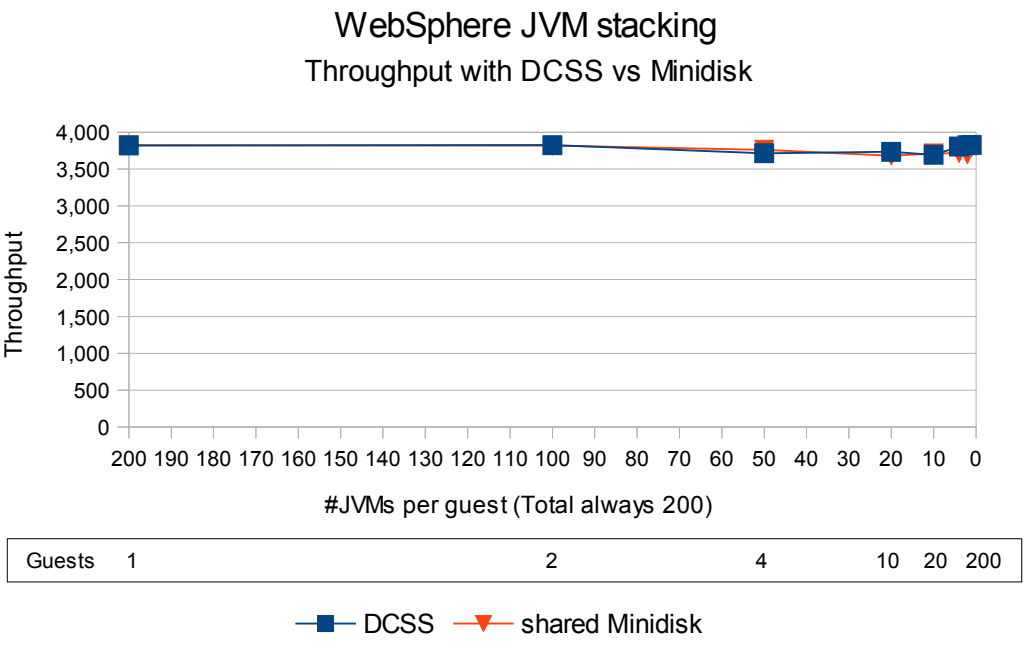
Where are the pages from the DCSS?

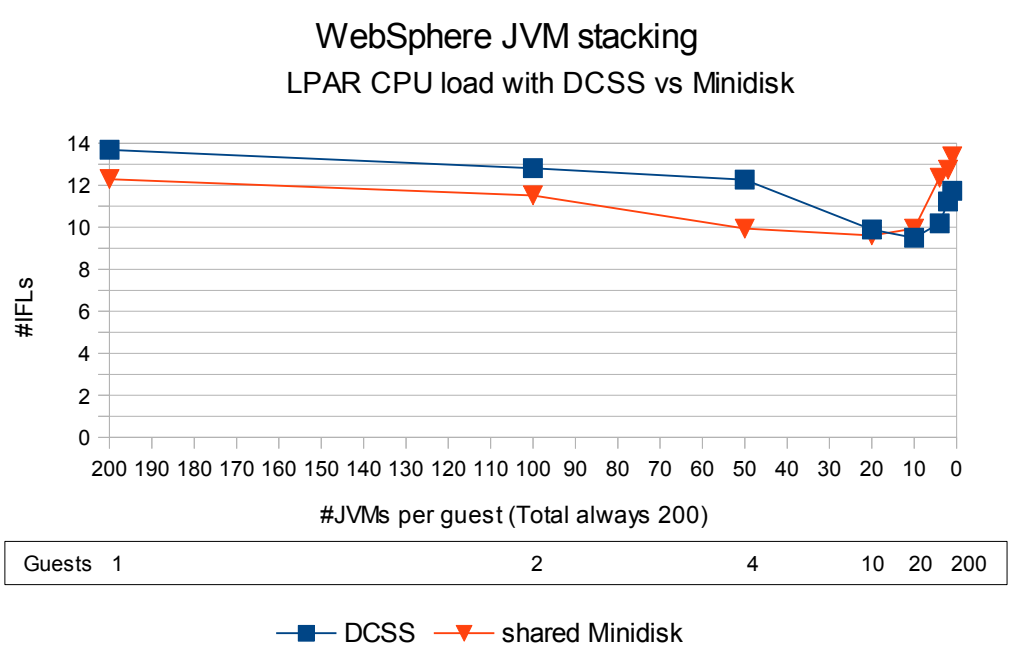


- **Looking at the page location from the pages of the DCSS**
  - At no point in time was the whole DCSS in storage, only the used pages
  - 400 MB are allocated in total

  These pages are used once at WebSphere start, but are now migrated because they are not used
  - Central storage contains in average about 10 MB, with a peak allocation of 50 MB
  - DCSS pages are never written to DASD!
- **We are saving at minimum 10 MB per guest**
- **This scales with the amount of guests: for 100 guests 1 GB memory and more are saved when using a DCSS**

Horizontal versus Vertical WAS Guest Stacking – DCSS vs Minidisk





DCSS or shared minidisks?

- **Impact on throughput is nearly not noticeable**
- **Impact on CPU is significant (and as expected)**
  - For small numbers of guests (1 – 4) it is much cheaper to use a minidisk than a DCSS (Savings: 1.4 – 2.2 IFLs)
  - 10 guests was the break even
  - With 20 guests and more, the environment with the DCSS needs less CPU (Savings: 1.5 – 2.2 IFLs)

Agenda

- Introduction
- z/VM Resource Manager (SRM)
- Recommendations for Memory Overcommitment on z/VM
- Large DCSS
- **Summary**

Memory Overcommitment Summary

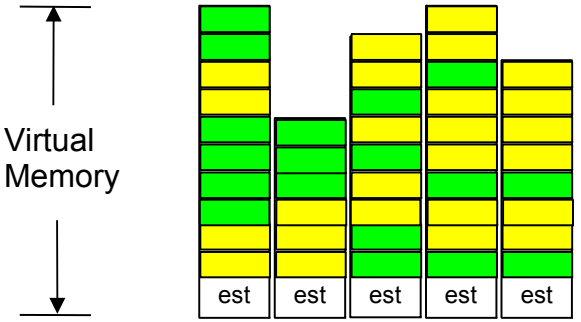
- **SRM settings**
  - Defaults prevent memory overcommitment for Linux guests (Q3 guests)
  - Adjust **LDUBUF** and **STORBUF** settings as required
- **Virtual memory = Physical memory** did not provide the best performance
- **Optimal memory settings are as long as there is no z/VM paging activity!**
- **In our environment the sum of all active Java heaps and allocated DB2 memory pools defined the minimum memory size**
  → This is really the lower limit, do not cross!
- **Monitor paging activity of your system:**
  - FCX113 User Paging Activity and Storage Utilization
  - FCX254 Available List Management
  - FCX135 Average User Wait State Data (→ %PGW)
- **More details in "Tivoli Provisioning Manager Version 7.1.1.1: Sizing and Capacity Planning"**
  - http://public.dhe.ibm.com/software/dw/linux390/perf/ZSW03168USEN.PDF
    Chapter 9. Memory overcommitment

DCSS - Summary

- **Time to save/ 1st load (saved DCSS)**
  - Do not make the DCSS larger than necessary
  - Consider using a separate guest just for this purpose
  - Moderate update frequency
- **Swap (not saved DCSS)**
  - Might be appropriate as swap device for memory peaks
- **Sharing Read Only data**
  - A DCSS is the ideal solution for that!
- **Sharing the software stack**
  - Installing the whole product in the DCSS provides a consistent source, allows to exchange the DCSS without dependency on the Linux guest
  - With 5 guests we already saw an advantage in performance using WebSphere and DB2 in the DCSS
  - With 20 and more guests the CPU cost is lower than a shared minidisk
- **More details at "Large Discontiguous Saved Segments (DCSS) Under Linux"**
  - http://public.dhe.ibm.com/software/dw/linux390/perf/ZSW03186USEN.PDF

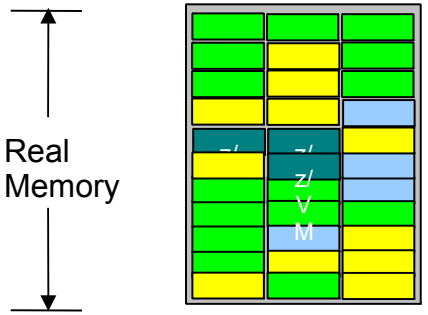
Backup

z/VM Memory Management



Active Virtual Memory Pages

Inactive Virtual Memory Pages

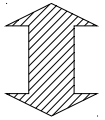


All Active Virtual Memory Pages **must** be covered with real memory.
Otherwise the application experiences a page fault when used

Inactive Virtual Memory Pages **might** be covered with real memory

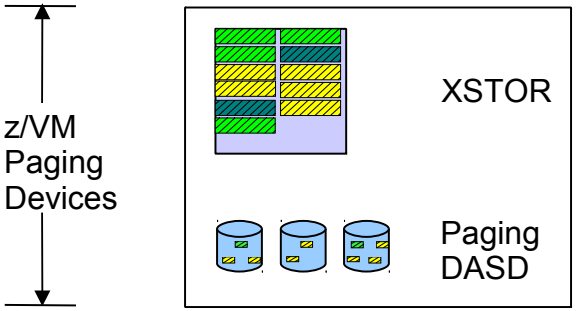
z/VM owned Memory Pages

Inactive Real Memory



When a guest experiences a page fault for an active page,
→ the required page must be brought into real memory **from** paging space

When all pages are in use and a free page is needed
→ a real memory page must be transferred **to** the paging space



Active Virtual Memory Pages on Paging Space

Inactive Virtual Memory Pages On Paging Space

z/VM owned Memory Pages on Paging Space