z/VM 6.3 Resource Overcommitment

Linux end-to-end Performance team:
Michael Johanssen: JOHANSSN@de.ibm.com
Dr. Juergen Doelle: juergen.doelle@de.ibm.com

Table of Contents

About this publication

This paper provides information about the performance of virtualized environments when operated under heavy workloads and with constrained resources. The analysis focuses on a z/VM® based system installation, with a set of virtual systems running different types of workloads under Linux on System z®, while the z/VM system was subjected to resource constraints. The impact of these z/VM resource constraints on the various workloads as well as on the z/VM system itself were investigated.

Another analyzed aspect was the efficiency of the z/VM paging subsystem when using EDEV-SCSI based paging devices, as opposed to ECKD™ based paging devices. The behavior of the new memory management introduced with z/VM 6.3, as opposed to the memory management used up to z/VM 6.2 was also inspected.

Note: The web-links referred in this paper are up-to-date as of October, 2014.

Introduction

The performance of various workload types running in a set of virtual systems under z/VM was evaluated. During these evaluations, the amount of processors and the amount of memory available to the z/VM host system were constrained.

The set of workloads was initially optimized in an unconstrained environment, and the resulting configurations were kept constant for all performed tests. This includes that the configuration of the virtual systems - such as the number of virtual processors, the relative shares and the amount of virtual memory - was kept constant as well.

For subsequent test executions, the number of real CPUs and the amount of real memory available to the z/VM host system were iteratively reduced, creating processor or memory shortages, or both, in the z/VM host system. The reductions were carried forward with each test iteration, until workload errors were encountered.

The resource shortages in z/VM had implications on the performance of the various workloads running under Linux® within virtual systems. Meaningful metrics were identified and monitored to determine the impact of the z/VM resource shortages on the workload performance, compared to the initial unconstrained environment.

Objectives

Find a list of objectives of this project in this topic.

- Evaluation of the performance implications caused by z/VM host CPU and memory constraints on different workload types running in virtual systems with Linux on System z. The following workloads types were investigated:
  - Database Business Intelligence (BI) workload
  - Transactional WebSphere® Application Server (WAS) workload
  - File system I/O workload
  - Java™ workload
  - Network workload
- Evaluation of the sensitivity of these different workload types against CPU and memory constraints at the host system
- Evaluation of the efficiency of
  - The z/VM memory subsystem when dealing with shortages of real memory
  - The z/VM scheduling subsystem when dealing with shortages of real processors
  - The z/VM paging subsystem when either EDEV-SCSI or ECKD based paging disks are used.

Notation conventions

A table within this section lists the notation conventions used in this paper in accordance to IEC 60027-2 Amendment 2.

Table 1. Notation conventions

Symbol	Full name	Derivation
KiB	kibibyte	2 ** 10 byte == 1024 byte
MiB	mebibyte	2 ** 20 byte == 1048576 byte
GiB	gibibyte	2 ** 30 byte == 1073741824 byte
KiB/s	kibibyte per second	2 ** 10 byte / second
MiB/s	mebibyte per second	2 ** 20 byte / second
GiB/s	gibibyte per second	2 ** 30 byte / second

Executive summary

From the perspective of an IT executive, read a short summary of the results and conclusions from the performance tests conducted in the study described in this paper. The provided information may help to make decisions about provisioning hardware and software resources for your enterprise.

The Large Memory Support and the HiperDispatch features introduced with z/VM 6.3 significantly improved the resource overcommitment behavior, as opposed to z/VM 6.2. In addition, the use of EDEV-SCSI devices for z/VM paging allowed substantially higher memory overcommitment levels when compared to using ECKD paging devices. z/VM 6.3 with EDEV-SCSI paging devices can be highly recommended for environments running at high memory overcommitment levels.

It is also substantiated that different types of applications behave very differently when operated in resource constrained virtualized environments. Especially Java based middle-ware reacted comparatively sensitive when z/VM real memory was constrained. On the other hand, applications performing mainly disk I/O or network intensive operations turned out to be robust with respect to z/VM memory overcommitment. A database BI workload was even able to increase its throughput in some z/VM resource constrained scenarios, probably due to its use of highly sophisticated caching and workload scheduling capabilities. When operating a z/VM environment with limited real memory and processor resources, a well-balanced mix of applications is recommended in order to achieve a uniform utilization of available system resources.

Summary

In this study we investigated the performance of a fixed set of workloads running in z/VM virtual systems while z/VM real memory and processors were progressively constrained. Scaling down the amount of z/VM real memory and the number of z/VM real CPUs caused an increasing ratio between virtual resources and real resources. The real resource shortages had implications on the performance of the workloads, with applications either waiting for getting scheduled on a CPU or for virtual memory being paged in from z/VM paging devices. The focus of the study was on the new z/VM 6.3 release that introduced major improvements in memory and processor management. Another aspect analyzed was the use of EDEV-SCSI devices for z/VM paging.

From an initial investigation performed in an oversized z/VM environment (25 real CPUs and 512 GiB real memory) it ensued that the workload set required about 21 real CPUs and 133 GiB real memory.

In subsequent iterations:

- The number of real CPUs was scaled down from 20 to 5,
- The amount of real memory was scaled down from 512 GiB to 64 GiB.

When scaling down the number of real CPUs, the overall workload throughput scaled about linearly, albeit the individual workloads scaled differently. This highlights an important aspect of the test setup: There was fierce competition for resources by the workload mix. Depending on the requirements of the individual workloads, it might happen that the performance of one workload degrades, while another workload continues unaffected or even profits. For example, one workload might wait for memory to be paged in and is thus unable to use CPU, while a second workload has low memory requirements and thus is able to make use of CPU not consumed by the first workload.

When scaling down the amount of real memory, the overall workload throughput stayed flat and even increased initially, until real memory was configured to be less than required as initially determined. Beyond that, the overall throughput decreased more or less sharply, depending on the z/VM environment. Here the performance of the z/VM 6.3 environment was superior to that of the z/VM 6.2 environment, and the use of EDEV-SCSI paging devices was superior to that of ECKD paging devices. Using the z/VM 6.3 environment and EDEV-SCSI paging devices, it was possible to scale z/VM real memory down to 64 GiB, while all workloads continued to operate in a stable manner, albeit with degraded performance. Note that 64 GiB real memory is less than half of the real memory that was required to operate the workload set without real memory constraints.

Another focus was on workload specific behavior:

- **Database BI workload within a large virtual system**
  The database BI workload was less impacted by z/VM real memory constraints. When in addition z/VM real CPUs were also constrained, the workload mostly was less affected, and in some cases even could increase its throughput. This is attributed to the sophisticated mechanisms used for caching and for asynchronously scheduling work that are commonly applied by modern databases. A noteworthy characteristic of the database BI workload was that it was operated in a virtual system exhibiting a very large virtual memory (300 GiB), whereof the workload on average actually used about 80 GiB. This is still about five times larger than the next largest virtual system used in the workload set.
- **Transactional WAS workload**
  The transactional WAS workload was executed in two differently sized virtual machine sets, imposing two workload levels - medium and high. Both levels behaved very similar. Once the z/VM real memory overcommitment exceeded a certain level a distinct impact was observed. The impact was less significant than that observed for the (pure) Java workload. A possible explanation for this behavior is that the transactional WAS workload used the gencon garbage collection policy that might have helped limiting the spread of memory accesses.
- **Java workload**
  The Java workload was also executed at two levels - medium and high - that behaved very similar. By trend, the z/VM workload also behaved similar to the transactional WAS workload. However, the setback of the throughput already occurred at higher z/VM real memory settings. A possible explanation is that this workload used the optthruput garbage collection policy that might have caused widespread memory accesses.
- **File system I/O workload**
  The file system I/O workload was executed in two variants, direct I/O and page cached I/O, and each variant was executed at two levels. The use of page-cached I/O versus direct I/O caused very different memory usage patterns. The page-cached variant used high amount of virtual memory (up to the complete virtual memory defined for its virtual system), causing relatively high page-read rates when z/VM real memory was constrained. Opposed to that the direct I/O variant had a very small memory footprint, but with intense memory usage (that is, high memory access rates). This caused most of the used virtual memory to be kept resident by z/VM, such that no page-read activity from the z/VM paging space occurred.

- **Network workload**
  The network workload was executed at two levels that behaved very similar. The network workload also behaved much like the direct I/O variant of the file system I/O workload, that is, the memory footprint was small but with intense usage, avoiding z/VM page-read activities for this workload.

Table 2 shows a categorization of various workloads with respect to their memory overcommit tolerance.

Remaining resident pages under memory pressure [1]				
		Low	**Medium**	**High**
Page read rate	**Low**	Network File system with direct I/O		
	Medium	DB2® transactional as WebSphere backend	Web Server Medium level file system I/O with page cache	high level file system with page cache
	High		Database BI WebSphere Application Server	Java
[1] Remaining resident pages under memory pressure classifies the workload with respect to the amount of resident pages that the workload still effects when running in a memory constrained z/VM environment, as opposed to an unconstrained z/VM environment.				

The workloads are characterized based on their page read rates and on the amount of virtual memory backed by real memory, when running in a memory constrained z/VM environment.

- The upper left corner represents workloads that are able to tolerate z/VM real memory constraints. Typically such workloads have a small memory footprint, but with intense memory usage, causing low z/VM page-read rates. When z/VM real memory is over-committed while executing a mixed workload set, these workloads are typically only moderately impacted.
- The lower right corner represents workloads that have a large memory footprint, along with widespread use of the memory, causing significant z/VM page-read rates. These applications have a high chance of showing performance impacts already when only small z/VM real memory constraints exist.

It is emphasized that the mixing of workloads turned out to be an essential factor that influences the potential for z/VM real memory overcommitment. A mixture of different kinds of workload enables some workloads to profit from other workloads when these are unable to make use of still available resources (such as for example CPU) while waiting for another essential resource (such as for example memory).

Hardware and software configuration

The hardware and software installation characteristics of the used test environment are illustrated. Also the configurations of the virtualized environments and the z/VM host system are described.

Figure 1 shows a symbolic representation of the used hardware and software, including the virtual system configurations.

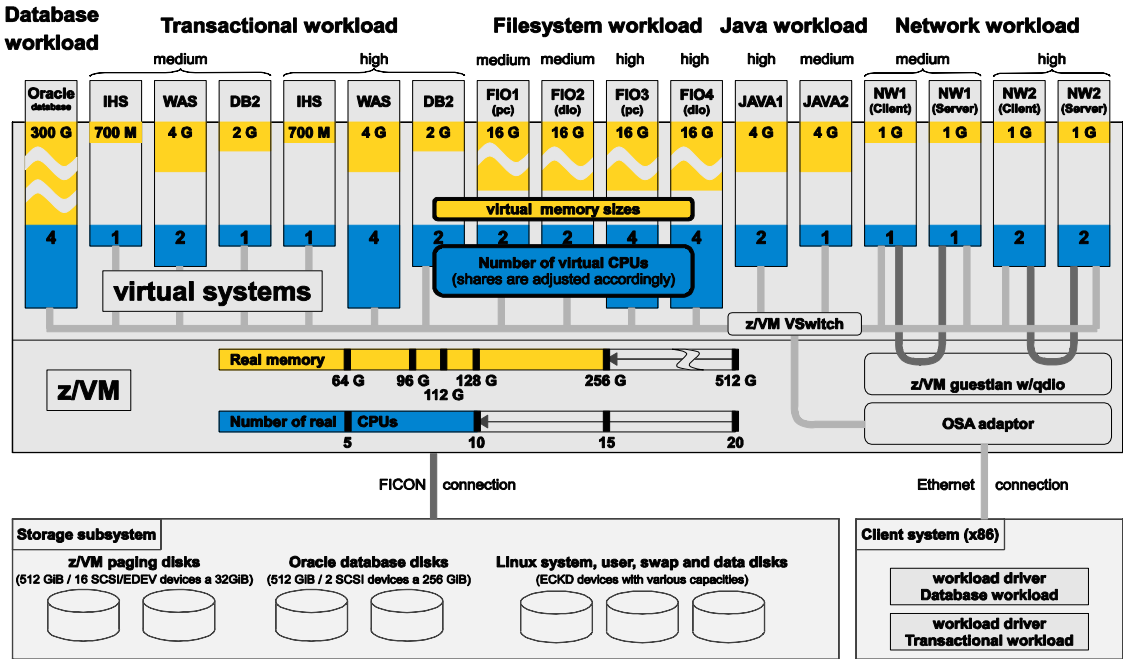

Figure 1: Test Environment

The top part of Figure 1 depicts virtual systems, detailing the amount of virtual memory (yellow) and the number of virtual processors (blue) configured for each virtual system. For details, see Virtual system configuration. Network connectivity for virtual systems is shown below their depiction. Note that these virtual configurations remained unchanged for all performed tests.

The middle part of Figure 1 depicts the amount of memory and the amount of processors configured for use by the z/VM host system. For details, see z/VM host configuration. The black marks depict the various amounts of memory and processors that were used for the performed tests.

The lower left part of Figure 1 depicts the storage resources in use by the z/VM host and by the virtual systems.

Note: SCSI as well as ECKD based storage resources were used.

The lower right part of Figure 1 shows the client system that was used to drive the database workload and the transactional WAS workload. In fact, the client system emulates workload requests that would normally result from large amounts of human interaction.

For each workload, one or more virtual systems running Linux on System z with following middle-ware or workload drivers were needed:

- Database BI workload
  - Oracle Database
- Transactional WAS workload

  - IBM Http Server (IHS)
  - WebSphere Application Server (WAS)
  - DB2 database
- File system I/O workload
  - fio
- Java workload
  - Java transactional workload application
- Network workload
  - uperf client (network client workload driver)
  - uperf server (network server component)

Hardware

The hardware components of the test environment consist of a host system, a storage server, and a client system server.

Host system

The host system was an IBM zEnterprise® 196 (z196), Model 2817-M66.

On that system, an LPAR was configured with processor and memory resources, as outlined in Table 3.

Table 3. LPAR processor and memory configuration

Resource	Initial	Reserved
Shared processors (CPs)	8	20
Memory	48 GiB	464 GiB

The use of shared processors was chosen because this is the typical configuration for customer environments, and because the vertical dispatch mode of the new z/VM HiperDispatch feature is only supported for shared processors. In order to ensure reproducible test conditions, this LPAR was the only active LPAR on the host system.

Note: Shared IFLs (Integrated Facility for Linux type processors) could have been used as well.

Access to ECKD disk storage was provided by four FICON Express8 8 Gbit channel cards that are connected to a FICON® switch.

Access to SCSI disk storage was provided by four FICON Express8 8 Gbit channel cards that are connected to an FCP switch.

Storage server

The storage server was an IBM System Storage® DS8800, Model 951 with eight host adapters that were each connected to the same FICON and FCP switches as the host system.

All essential ECKD and SCSI disks were configured on that storage server.

Client system server

The client system server was an IBM Flex System™, Model 8737-R2G.

Table 4. Client system configuration

Resource	Initial
Processors (CPUs)	2 Intel Xeon processors E5-2690 operating at 2.9 GHz
Memory	250 GiB DDR3
Network connectivity	10 GbE network card

Software

View information about the software components and benchmarks used in the described test scenarios.

Table 5. Software elements

Operating systems	
Hypervisor System z	IBM z/VM Version 6 Release 3.0, service level 1401 (64-bit) Product number 5741-A07 ▪ Installed in System z LPAR
Guest operating system System z	Linux on System z SUSE Linux Enterprise Server 11 (390x), SLES 11, SP3 ▪ Installed in z/VM virtual systems
Operating system client system	Linux on IBM System x® SUSE Linux Enterprise Server 11 SP2 (x86_64) ▪ Installed on System x (x86_64) server
Middle-ware	
Oracle Database	Oracle Database 11*g* Enterprise Edition Release 11.2.0.3.0 - 64bit ▪ Installed under Linux on System z
WebSphere Application Server (WAS)	IBM WebSphere Application Server Network Deployment, 8.5.5.1 Build Number: cf011341.03 Build Date: 10/18/13 ▪ Installed under Linux on System z
WebServer	IBM HTTP Server (IHS) Server Server version: IBM_HTTP_Server/8.5.5.1 (UNIX®) Apache version: 2.2.8 (with additional fixes) Server built: Jul 11 2013 18:01:24 Build level: IHS85/webIHS1327.01 ▪ Installed under Linux on System z
DB2	DB2 v10.5.0.2, s131001, IP23538 and Fix Pack 2. ▪ Installed under Linux on System z
Test suites	
fio	fio 2.1.7 committed 2014-03-31, for details see the *fio readme* fio readme ▪ Installed under Linux on System z
DayTrader	IBM WebSphere Application Server Samples DayTrader 3.0.7 (No Caching) Full EE6 Spec Compliant for WebSphere 8.5.5.0 Date: 20130925 For details, see *Daytrader - a more complex application* ▪ Installed under Linux on System z
SwingBench	Version 2.5.0.909 For details, see SwingBench 2.2 Reference and User Guide. ▪ Installed under Linux on System x on System x client system
uperf	Version 1.0.4 For details, see *uperf - A network performance tool* ▪ Installed under Linux on System z
Workload generators	
Workload simulator	WebSphere Studio Workload Simulator 5655-I14 Version 03309L

z/VM host configuration

The z/VM host system was configured to exploit the set of processors and the amount of memory that were configured as initial for the hosting LPAR.

If required by a particular test execution, these resource sets were modified during the test execution setup phase by activating or deactivating processors, and/or by claiming additional memory from the LPAR's reserved memory, as follows:

- CPUs were activated/deactivated as required, by means of the `vary on/off processor <nn>` command.
- Additional memory was made available by means of the `set storage` command.

This approach makes it possible to dynamically control the amount of CPUs and memory used by the z/VM host for a particular test execution, without having to change the activation profile defined in the hardware management console (HMC) for the LPAR.

For z/VM 6.3, the documentation does not recommend the use of expanded storage. Thus in this case all memory required for a test case was defined as central storage, and no expanded storage was defined.

For z/VM 6.2, the documentation recommends the use of expanded storage. Thus in this case 2 GiB of the memory required for a test case were defined as expanded storage, while the remaining memory was defined as central storage.

Two variants were configured for the z/VM paging devices:

- **EDEV-SCSI variant:**
  The EDEV-SCSI variant was composed of 16 EDEV paging devices based on SCSI devices configured in the storage server. Each SCSI device had a size of 32 GiB, resulting in a combined paging area of approximately 512 GiB. The value of the ATTRIBUTE parameter on the z/VM EDEV configuration statements was set to 2107. This is the recommended value for SCSI disks residing on DS8000® like storage servers.
  When using EDEV-SCSI devices for paging, z/VM is able to issue overlapping I/O operations for each EDEV-SCSI device (for details, see *z/VM 6.3 Performance Considerations*). In other words, I/O operations for a particular EDEV-SCSI paging device are parallelized.
- **ECKD variant:**
  The ECKD variant was composed of 25 ECKD 3390 devices that were configured on the storage server. Each 3390 device had a size of 30051 cylinders. The format established by the z/VM format utility CPFMTXA established 180 pages per 3390 cylinder, resulting in 20.6 GiB per device. All 25 devices composed a paging area of approximately 516 GiB.
  z/VM (as of z/VM 6.3) does not support PAV (or HyperPAV) for paging volumes. This means that only one I/O operation at a time can be in progress for a particular paging device.

On the storage server, the SCSI and the ECKD disks were organized in striped storage pools from both internal servers.

Sufficient spool space was configured in the form of ECKD devices located on the same storage server.

An additional large number of 3390 model 9 and model 27 DASDs were configured on the storage server. These DASDs were assigned to the virtual systems running Linux in the form of full-pack minidisks, in order to serve as Linux root and paging devices. Furthermore, relating to the 3390 DASDs, ranges of HyperPAV devices were also configured on the storage server. These HyperPAV devices were directly attached to z/VM, in turn enabling z/VM to support **virtual** HyperPAV devices for the full-pack minidisks assigned to the virtual systems.

The z/VM system was connected to an internal TCP/IP network via an OSA Express2 adapter. A z/VM Virtual Switch (VSWITCH) was configured connecting to that OSA adapter, enabling the virtual systems running Linux to use the VSWITCH for external network access.

z/VM monitor event data collection and z/VM monitor sample data collection were selectively enabled for the processor, storage, user, I/O, and network domains.

Virtual system configuration

Read the information about the configuration of the virtual systems used in the test scenario.

All virtual systems running Linux had the following in common:

- One of the following disk configurations*:*
  - One full-pack minidisk on a FICON attached 3390 Model 27 DASD, formatted with two partitions: one partition for the root file system, and one as a Linux swap device.
  - Two full-pack minidisks on a FICON attached 3390 Model 9 DASD, formatted with one partition each. The first disk was used for the root file system, the second disk was used as a Linux swap device.
  - Note: Even though Linux swap devices were configured, the virtual machines, the Linux systems and the workloads were configured such that Linux swapping did not occur.
- For each virtual full-pack minidisk, four virtual HyperPAV devices were configured in the z/VM directory definition for the virtual system containing the minidisk, as follows:

  ```
  USER LE2EPxxx
  ...
  COMMAND DEFINE HYPERPAVALIAS aaa1 FOR base bbbb
  COMMAND DEFINE HYPERPAVALIAS aaa2 FOR base bbbb
  COMMAND DEFINE HYPERPAVALIAS aaa3 FOR base bbbb
  COMMAND DEFINE HYPERPAVALIAS aaa4 FOR base bbbb
  ...
  MDISK bbbb 3390 DEVNO bbbb
  ```

  In this example, bbbb is the device number used for both the real disk and the virtual full-pack minidisk, and aaa1-4 are the devices numbers of the virtual HyperPAV devices.
- A virtual NIC was attached to a z/VM Virtual Switch (VSWITCH), connected through an OSA adapter to an internal network.

Table 6 lists the virtual system configurations that were used throughout the tests.

Table 6. Virtual system configuration variants (constant for all performed tests)

Workload				# CPUs	Memory	Additional resources
#	Type	Level	Component or variant			
1	Database		Database server	4	300 GiB	2 * 256 GiB SCSI disks (DATA) 3 * 10 GiB SCSI disks (LOG)
2	Transactional	medium	IHS	1	700 MiB	
3			WAS	2	4 GiB	
4			DB2	1	2 GiB	
5		high	IHS	1	700 MiB	
6			WAS	4	4 GiB	
7			DB2	2	2 GiB	
8	File system	medium	page cached	2	16 GiB	4 * 3390 Model 9 (fio data files)
9			direct I/O	2	16 GiB	4 * 3390 Model 9 (fio data files)
10		high	page cached	4	16 GiB	4 * 3390 Model 9 (fio data files)
11			direct I/O	4	16 GiB	4 * 3390 Model 9 (fio data files)
12	Java	high		2	4 GiB	
13		medium		1	4 GiB	
14	Network	medium	client	1	1 GiB	1 VNIC to QDIO based guest LAN
15			server	1	1 GiB	1 VNIC to QDIO based guest LAN
16		high	client	2	1 GiB	1 VNIC to QDIO based guest LAN

Assignment of relative shares to virtual systems

In z/VM, virtual systems are assigned *relative shares* that control the allotment of resources such as CPU, real storage, or paging capacity, when these resource are constrained at the z/VM level (for details, see *z/VM 6.3 Performance*). *Relative* means that when virtual systems compete for a constrained resource, the resource is allotted by factoring in the relative shares assigned to the competing virtual systems. For example, if a virtual system **A** was assigned twice the relative share assigned to a virtual system **B**, then **A** receives twice as much resource as **B**.

It is important to realize that the treatment of shares for the resource type CPU does not depend on the amount of virtual CPUs assigned to a virtual system. For example, if virtual system **C** had one virtual CPU, and virtual system **D** had four virtual CPUs assigned, but both virtual systems had the same relative share assigned, then, - when z/VM runs into a CPU constrained situation -, both systems would receive the same amount of CPU, irrespective of their number of virtual CPUs. For example, if in this situation z/VM had two CPUs available for dispatching virtual system work, then on average both virtual systems would receive a processing power of one CPU. If all virtual CPUs were active all the time, then **C**'s only virtual CPU would receive a processing power of one real CPU, but each of **D**'s four virtual CPUs would receive a processing power of only 0.25 real CPUs.

In order to achieve a distribution of processing power with respect to the amount of virtual CPUs defined for a virtual system, in the test environment, virtual systems were assigned relative shares with their number of virtual processors factored in. For example, a virtual system with one virtual CPU was assigned a relative share of 100, a virtual system with two virtual processors was assigned a relative share of 200, and so forth. With this approach, in the situation described in the previous paragraph, the one virtual CPU of virtual system **C** would then receive a processing power of 0.4 CPUs, and each of virtual system **D**'s virtual CPU would also receive a processing power of 0.4 CPUs.

However, a virtual CPU can never consume more than one real CPU, regardless of the share. So if z/VM in the same situation had six real CPUs available for dispatching virtual system work, then each virtual CPU would receive the processing power of one real CPU, and the equivalent of one real CPU would remain more or less unused.

Test approach
Read the contained information about the design of the tests used in this study.

Aspects like the test workloads, the configurations, and the procedure are discussed. Also, the metrics used to evaluate the performance of the test workloads are outlined in this topic.

Test workloads
Different types of workload have been used in the performed tests.

The listed sections describe the used types of workloads in more detail:
- Database BI workload
- Transactional WAS workload
- File system I/O workload
- Java workload
- Network workload

Database BI workload

SwingBench was used for the database BI workload. SwingBench is a free load generator (and benchmark) designed to stress-test an Oracle Database. For more information, see:
http://www.dominicgiles.com/swingbench.html

SwingBench comes with four customized benchmarks: OrderEntry, SalesHistory, CallingCircle, and StressTest. For the overcommitment project discussed in this publication, the SalesHistory benchmark was selected because it emulates a business intelligence (BI) workload.

SwingBench is driven by an external client that communicates with the Oracle Database. In our case, the client was installed on a x86-64 server under Linux. The Oracle Database was installed within a virtual system running Linux on System z. For details on the virtual system, see Virtual system configuration.

Communication between client and database was established by means of JDBC type IV connectors over an IP network.

The Oracle Database was configured using ASM as disk storage management tool for a data disk group and a log disk group:
- Two 256 GiB SCSI data disks within a data disk group
- Three 10 GiB SCSI log disks within a log disk group

Table 7 details the parameters that are set for the Oracle Database.

Table 7. Oracle Database parameters

Parameter	Name	Value
SGA size	sga_target	200 GiB
PGA size	pga_aggregate_target	45 GiB

The SwingBench shwizard tool was used to create the database elements required for the SalesHistory workbench such that the total size of disk space used for the SalesHistory execution was about 300 GiB.

The SalesHistory workload was customized for producing specific loads on memory and I/O resources, as shown in Table 8.

Table 8. SwingBench SalesHistory workload parameters

Parameter	Name	Value
number of SwingBench users	-uc	16

Database BI workload performance metric: The SalesHistory workload randomly selects the transactions it performs from a set of transaction types of varying complexity. For that reason, the Database BI workload performance metric was defined as a *weighted transaction rate* (transactions per second), using weight factors that were determined from the execution times that the various transaction types achieved during the reference execution. For each test execution, the weighted transaction rate then was determined, taking the transaction rates reported separately for each transaction type by the workload driver, factoring in the respective transaction type specific weight factor and summing up these weighted transaction rates.

Transactional WAS workload

For the transactional WAS workload, an adaptation of the DayTrader workload to the WebSphere Application Server (WAS) was used.

DayTrader is an Open Source benchmark application emulating an online stock trading system. Users can log in and view their accounts or portfolios. In the Quotes/Trade section they can buy or sell stock shares out of their portfolios. The benchmark application is currently hosted by the Apache Geronimo project and was originally developed by IBM as the WebSphere Trade performance benchmark. This benchmark was donated to the Apache Geronimo project in 2005.

DayTrader is an end-to-end Java Enterprise Edition (J2EE) web application composed of several Java classes, Java Servlets, Java Server Pages, Web Services, and Enterprise Java Beans (EJB). Thus, it is an ideal benchmark application for measuring the scalability and performance of a J2EE application server like IBM WebSphere Application Server (WAS). The presentation layer is based on Java Servlets and Java Server Pages, whereas the back end business logic uses Java database connectivity (JDBC), Java Message Service (JMS), EJB, and Message Bean techniques. For our tests, the Trade Database was hosted on a DB2 database server and was accessed via JDBC from the application server. For more details on the DayTrader benchmark, see *DayTrader - a more complex application*.

Transactional WAS workload variants: The transactional WAS workload was executed at two levels, medium and high, using two suites of z/VM virtual systems. Each suite was composed of three virtual system running Linux on System z, where each virtual system had one of the following components installed:

- IBM Http Server (IHS)
- WebSphere Application Server (WAS)
- DB2 database server (DB2)

For details on the virtual systems, see Virtual system configuration. For details on the software components, see Software.

The difference between the medium and the high level is shown in Table 9.

Table 9. Transactional WAS workload parameters

Level	Component	# virtual CPUs
medium	WAS	2
	DB2	1
high	WAS	4
	DB2	2

WebSphere Application Server configuration: WebSphere Application Server was installed using the Network Deployment (ND) variant. Table 10 details the parameters that were set for both levels of the WebSphere Application servers.

Table 10. WebSphere Application Server parameters

Parameter	Name	Value
Min. ORB pool size	minOrbPool	10
Max. ORB pool size	minOrbPool	50
Min. WebContainer pool size	minWebPool	50
Max. WebContainer pool size	minWebPool	50
Min. Default pool	minDefaultPool	5
Max. Default pool	minDefaultPool	150
Keep alives enable	keepAliveEnabled	TRUE
Min. JVM java heap size	minHeap	2048
Max. JVM java heap size	maxHeap	2048
Java garbage collection policy	default WAS 8.5 with Java 7	gencon
Min. TradeDS pool	minTradeDSPool	200
Max. TradeDS pool	maxTradeDSPool	300
Min. Broker pool	minBrokerPool	1
Max. Broker pool	maxBrokerPool	150
Min. Streamer pool	minStreamerPool	1
Max. Streamer pool	maxStreamerPool	150

DayTrader workload configuration: Table 11 lists the parameters applied for both levels of the DayTrader workload.

Table 11. DayTrader workload parameters

Parameter	Name	Value
run time mode	runTimeMode	1 (direct JDBC)
order processing mode	n/a	Asynchronous_2-Phase
max. number of users	minOrbPool	15000
max. number of quotes	minWebPool	10000

Workload driver configuration: A workload driver was used to emulate user interactions that generate load against the DayTrader application. The workload driver was customized to run http transactions against the DayTrader environment. These transactions performed changes on the DayTrader database by means of Web Services provided through the WebSphere environment.

The workload driver was installed on a client system server. For details on the client system server, see Client system server. For details on the software components, see Software.

Table 12 lists the workload driver parameters that were applied:

Table 12. Workload driver parameters

Parameter	Name	Value
think time [ms]	thinktime	500
number of simultaneously running clients	-c	500
element delay percent	-e	0
dynamic cookies	-D	on

Transactional WAS workload performance metric: The *transaction rate* (transactions per second) determined over the complete workload execution time was used as metric for the performance of the transactional WAS workload.

File system I/O workload

For the file system I/O workload, the fio tool was used. fio is an I/O tool intended to be used both for benchmarking, and for stress/hardware verification. For more information, see the fio readme file.

fio has support for various types of I/O engines (such as sync or libaio), I/O priorities, throughput, forked or threaded jobs, and much more. It can work on block devices as well as files. fio displays all sorts of I/O performance information.

File system I/O workload variants: For our tests, four variants of the file system I/O workload were executed within separate Linux on System z instances running within z/VM virtual systems. For details on the virtual systems, see Virtual system configuration.

For each variant, the file system I/O workload was varied such that specific loads on memory and I/O resources were produced. The file system workload parameters are listed in Table 13.

Table 13. File system I/O workload parameters

Parameter	Name	Values			
		medium level		high level	
		page-cached	direct I/O	page-cached	direct I/O
enable direct I/O	direct	0	1	0	1
type of I/O engine	ioengine	sync	libaio	sync	libaio
I/O depth	iodepth	1	4	1	4
Number of fio jobs	numjobs	8		16	
Number of files	nrfiles	1			
file size	filesize	1536 MiB			
block size	bs	32 KiB			
type of I/O pattern	rw	randrw			
percentage of read	rwmixread	80			

Types of I/O operations: The page cache is memory that is maintained by the Linux kernel for the purpose of caching data related to user process I/O operations. If the use of the page cache is requested with a particular I/O operation (the default case), then the Linux system transparently caches I/O related data in the page cache. With write operations, the Linux system decides when to transfer data from the page cache to the respective file. For example, I/O operations may be queued to be performed when respective I/O and system resources are available. Of course, the use of the page cache implies a higher memory usage. On the other hand, if direct I/O is used, then the data transfer by the fio user process goes directly from the application buffers to the data files, avoiding the use of the page cache.

The direct parameter controls the use of the page cache:
- direct=1 requests fio to perform direct I/O operations
- direct=0 requests fio to perform page cached I/O operations

In addition, for direct I/O operations, the use of native Linux asynchronous I/O was requested by means of the ioengine parameter, and the number of parallel I/O operations per process was limited by means of the iodepth parameter.

Files used for I/O operations: The number of files in use by a particular fio workload was controlled by means of the numjob and the nrfiles parameters.

For the medium level file system I/O workloads, eight fio processes were used. For the high level file system I/O workloads, sixteen fio processes were used. In any case, each process was requested to only use one file. The distinction between medium and high levels was that the medium level workload operated on eight files, while the high level workload operated on sixteen files.

The characteristics of the files were set to a file size of 1536 MiB, and a block size of 32 KiB.

The last two lines in Table 13 show that for all fio executions random read and write operations were requested by means of the rw parameter, with the rwmixread parameter requesting a mixture of 80 % read and 20 % write operations.

File system I/O workload performance metric: The sum of the fio read rate and the fio write rate (both in byte per second as reported by the workload driver) for all fio processes, determined over the complete workload execution time, was used as metric for the performance of the file system I/O workload.

Java workload

For the Java workload, a Java program was used that executed a number of parallel threads, with each thread performing the same kind of transactions.

For our tests, two levels, medium and high, of the Java workload were executed, each within a separate Linux on System z operating system instance running within a z/VM virtual system. For details on the z/VM virtual systems, see Virtual system configuration.

The Java workload itself was identical for both levels. However, for the medium level workload, one virtual processor was used, whereas for the high level workload, two virtual processors were used.

The Java workload parameters are listed in Table 14.

Table 14. Java workload parameters

Parameter	Name	Values
Java heap size (fixed)	max_heap	3200 MiB
	init_heap	3200 MiB
Java garbage collection algorithm	-Xgcpolicy	optthruput

Java workload performance metric: The sum of the transaction rate (transactions per second) for all processes, determined over the complete workload execution time, was used as metric for the performance of the Java workload.

Network workload

For the network workload, the uperf network performance tool was used. For details, refer to *uperf - A network performance tool.*

Client and server part were spread into multiple processes, with each pair of processes exchanging messages through separate TCP/IP network connections.

The network workload parameters are listed in Table 14.

Table 15. Network workload parameters

Parameter	Name	Values
Number of processes	n/a	250
Protocol	n/a	tcp

Network workload performance metric: The average data exchange rate [MiB/s] as reported by the workload driver aggregated for all processes and over the complete workload execution time was used as metric for the performance of the network workload.

Combined workload set

The combined workload set is the combination of all levels of the individual workloads:

- Database BI workload
- Transactional WAS workload (medium and high level)
- File system I/O workload (page-cached and direct I/O, each at medium and at high level)
- Java workload (medium and high level)
- Network workload (medium and high level)

Test configurations

A variety of distinct z/VM configurations were used for the tests. Each z/VM configuration is characterized by a unique combination of the amount of z/VM real memory, the amount of z/VM real processors, the type of paging device, and the z/VM release level.

In order to reduce textual replications within this document, configurations were assigned configuration IDs (CIDs), according to the following identification scheme:

```
<CID> = <ZVMmem>/<ZVMcpu>
```

where:

`<ZVMmem>` is the amount of real memory configured for z/VM, in GiB
`<ZVMcpu>` is the number of real processors configured for z/VM

For example, if z/VM was configured with 128 GiB of real memory and with 15 real CPUs, then the CID would be: 128/15.

Table 16 lists the CIDs for various z/VM CPU and z/VM memory configurations.

Table 16. Configuration IDs (CIDs) for various z/VM CPU and z/VM memory settings

	512 GiB	256 GiB	128 GiB	112 GiB	96 GiB	64 GiB	48 GiB
20 CPUs	512/20	256/20	128/20	112/20	96/20	64/20	48/20
15 CPUs	512/15	256/15	128/15	112/15	96/15	64/15	48/15
10 CPUs	512/10	256/10	128/10	112/10	96/10	64/10	48/10
5 CPUs	512/5	256/5	128/5	112/5	96/5	64/5	48/5

Test procedure

Read a list of steps that were performed for each test.

- IPL z/VM system
- Reconfigure z/VM number of active processors and amount of memory
- Start the combined workload set (see Combined workload set), that is, all levels and all components of all workloads, within respective virtual systems
- After 5 minutes, start z/VM monitoring
- After 50 minutes, stop z/VM monitoring
- After 5 minutes, terminate workloads
- Wait until all workloads have terminated
- Collect performance data

During each test, various Linux performance data and application related performance data was collected. Performance data from the z/VM monitor was collected during the central 50 minutes of each test execution, thus skipping peak loads occurring particularly during workload start-up. After the 60 minute execution phase, all workloads were terminated, but allowed additional time to complete eventually still running activities. Particularly the Database BI workload is characterized by long running transactions that could run for more than 20 minutes.

Test metrics

The performance of the various workloads and of z/VM were evaluated using the metrics outlined in this topic.

Workload performance metrics

All workload performance metrics are not presented as absolute values, but as the relative values with respect to the reference result.

Database BI workload relative throughput: relative throughput of the Database BI workload with respect to the reference result. For details, see Database BI workload.

File system I/O workload relative throughput: relative throughput of the File system I/O workload with respect to the reference result. For details see **File system I/O workload**.

Java workload relative throughput: relative throughput of the Java workload with respect to the reference result. For details, see Java workload.

Network workload relative throughput: relative throughput of the Network workload with respect to the reference result. For details, see Network workload.

Transactional WAS workload relative throughput: relative throughput of the transactional WAS workload throughput with respect to the reference result. For details, see Transactional WAS workload.

Overall workload relative throughput: combined relative throughput of all workloads. This gross metric is the sum of the relative throughput of each of the individual workload variants, divided by the number of workload variants.

z/VM performance metrics

When applicable, the z/VM metrics are ascribed to values presented in Performance Toolkit for VM™ screens. In this case, the origin of a metric is listed by naming the Performance Toolkit for VM Toolkit report identification, and the name of the origin item in the report. Occasionally, multiple reports or items are listed if more than one context (such as general versus virtual machine specific) applies. For details about the Performance Toolkit for VM, see z/VM 6.3 Performance Toolkit Reference.

For better comparability, the preferred units used for metrics in this document are

- Based on bytes for memory,
- Based on bytes/s for memory rates,
- Based on processors for CPU loads.

If necessary, values presented in different units by the *Performance Toolkit for VM* were converted to these units.

CP attributed to guest CPU load: the CPU load spent for processing z/VM control program code that relates to individual virtual systems, such as simulating the execution of a privileged instruction for a virtual system. CP attributed to guest CPU load is determined by subtracting the emulation CPU load from the CPU load.

CP system CPU load: the CPU load spent for processing z/VM control program code that cannot be attributed to individual virtual systems. The CP system CPU load for the whole z/VM system is reported in [FCX100:%SYS] (separately for each processor and processor type).

CPU load: the processing power consumed by one or more virtual systems, measured in fractions of real processors. Each virtual processor can consume a maximum of one real processor. The CPU load for the whole z/VM system is reported in [FCX100:%CPU] (separately for each processor and processor type). The CPU load of individual virtual systems is reported in [FCX112:%CPU].

DPA size: the size of the z/VM dynamic paging area (DPA). The DPA is the region of real memory that z/VM maintains for resident user pages. It accounts for the by far largest part of real memory used by z/VM. The DPA size is reported in [FCX103:Total DPA size].

emulation CPU load: the CPU load spent for processing virtual system workload, that is, while program code directly originating from virtual systems is executed. The emulation CPU load is reported for the whole z/VM system in [FCX100:%EMU] (separately for each processor).

instantiated memory: the amount of memory in use by a virtual system. It is composed as follows:

- Memory accessed (read or written) by code running in the virtual system
- Minus memory that was released back to CP, and not accessed again

The instantiated memory is usually smaller than the sum of resident memory and stored memory, because it counts memory pages kept in both places as one page. The instantiated memory of individual virtual systems is reported in [FCX292:Inst].

page rate: the sum of the page read rate and the page write rate. May be shown for one or more virtual systems, or for the whole z/VM system.

page read rate: the amount of data read per second from the z/VM paging store in MiB/s. May be shown for one or more virtual systems, or for the whole z/VM system. The overall z/VM page read rate is reported in [FCX103:Page read rate]. The virtual system page read rate is reported in [FCX290:<Movement/s> Reads] for z/VM 6.3, and in [FCX113:<Page Rate> Reads] for z/VM 6.2 and 6.3.

page write rate: the amount of data written per second to the z/VM paging store. May be shown for one or more virtual systems, or for the whole z/VM system. The overall z/VM page write rate is reported in [FCX103:Page write rate]. The virtual system page read rate is reported in [FCX290:<Movement/s> Write] for z/VM 6.3, and in [FCX113:<Page Rate> Write] for z/VM 6.2 and 6.3.

real CPU count: the number of CPUs configured for z/VM. For the tests described in this paper, central processors (CPs) were configured at the LPAR level (see Hardware), but Integrated Facilities for Linux (IFLs) could have been used as well. The count of real CPUs is reported in [FCX304:CPU], separately for each processor type.

real memory: the amount of memory configured for z/VM. The real memory size configured for z/VM is reported in [FCX103:Total real storage].

resident memory: the amount of memory backed in the z/VM DPA for a particular virtual system. The resident memory of individual virtual systems is reported in [FCX292:T_All].

stored memory: the amount of memory stored in the z/VM paging store for a particular virtual system. The stored memory of individual virtual systems is reported in [FCX292:AUX].

virtual memory: the amount of virtualized memory defined for a virtual system. The virtual memory size of individual virtual systems is reported in [FCX292:Base Space Size].

Overcommitment considerations

Read the provided detailed information about over-committing z/VM real memory and z/VM real CPUs. You can understand why overcommitment of z/VM real memory and real CPU is technically possible, and what the implications are. Readers only interested in the test results might skip this topic.

Memory overcommitment

When over-committing memory in a multi-layered virtual environment, the various tiers of memory virtualization must be considered.

Figure 2 illustrates the two tiers of memory virtualization existing in the test environment. The two tiers are indicated on the left side of Figure 2.

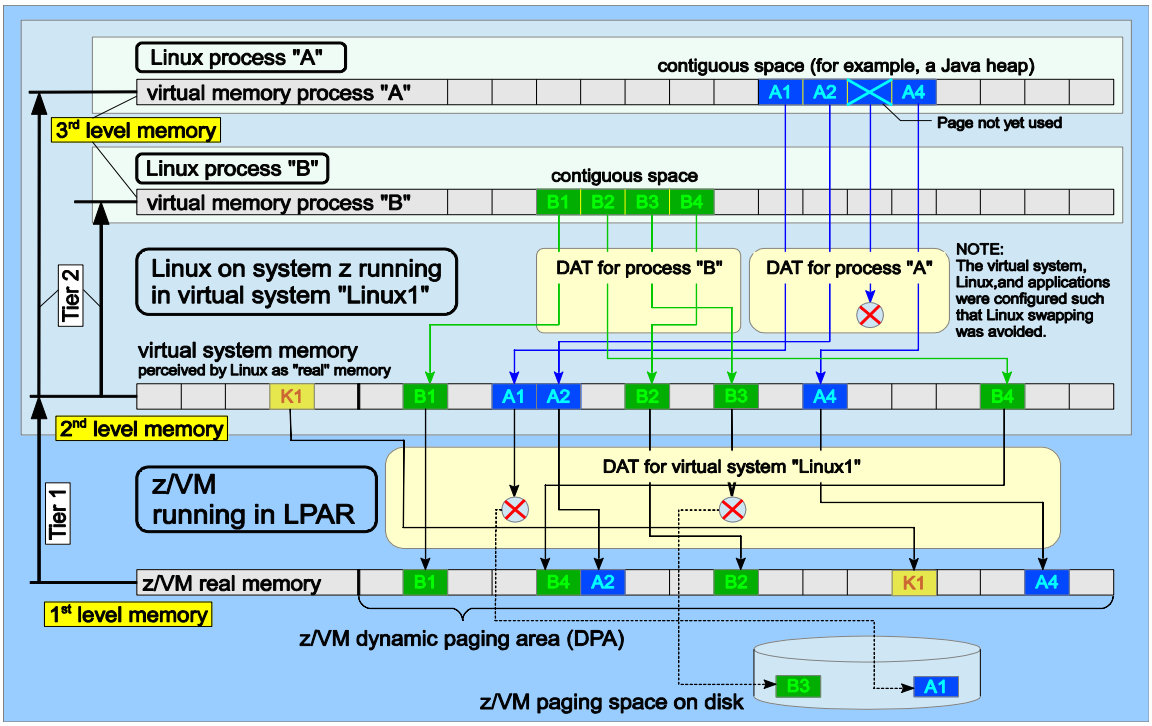

Figure2: Two tiers of memory virtualization

The lower part of Figure 2 shows the first tier of memory virtualization that z/VM establishes for the virtual memory of a virtual system. The first tier virtualizes 1st level memory - the memory configured for the LPAR - to 2nd level memory - the virtual memory provided by z/VM to a virtual system.

The upper part of Figure 2 shows a second tier of memory virtualization that Linux on System z establishes for its processes. The second tier virtualizes 2nd level memory - the virtual memory available within the virtual system - to 3rd level memory - the virtual memory provided by Linux to its processes.

Note that the second tier of memory virtualization is established by Linux with respect to 2nd level memory that - while Linux perceives it as real memory - in fact is already virtualized by z/VM as the virtual system's memory. When the word real is subsequently used in this section with respect to memory, it designates memory that is at the bottom of a memory virtualization tier.

In order to focus on the central concepts of memory virtualization, Figure 2 is simplified in many aspects. For example, it does not account for sharing real memory that backs identical virtual memory regions of two or more consumers (such as shared code). Another simplified aspect are elements of z/VM memory virtualization aiming at improving memory virtualization performance in layered environments, such as the use of shadow DAT tables.

Furthermore, communication mechanisms that exist between Linux and z/VM for the purpose of coordinating the effective use of memory are not shown.

Dynamic address translation (simplified)

Dynamic address translation (DAT) is a mechanism that allows mapping virtual memory to real memory. The **provider** of virtual memory is typically a control program such as the z/VM control program, or the Linux kernel. The **consumer** of virtual memory is a System z processor that is executing some kind of workload and that is running with DAT enabled. Enabling DAT causes the processor's memory accesses (such as fetching instructions and reading from or writing to memory) to take place with respect to virtual memory.

The provider of virtual memory must ensure that each page of virtual memory that is accessed by the consumer is mapped to a page frame in real memory. The mapping from virtual to real memory is maintained in the form of a hierarchy of DAT tables. For simplicity, we limit the discussion here to the lowest element in that hierarchy, which is called a *page table*.

Page tables contain *valid* entries for virtual pages that have an assigned corresponding *page frame* in real memory, and *invalid* entries for pages that do not have a corresponding page frame assigned. The reason for invalid entries is that the provider - for example when facing a situation where insufficient real memory is available - can decide to store the content of certain consumer pages on persistent store (such as a paging or swap disk), and invalidate the corresponding page table entry. This enables the provider to then use the respective page frame for other purposes, such as backing a page for another consumer.

In case an invalid entry is encountered while a consumer executes a program accessing the corresponding virtual page, program execution is interrupted and the provider is notified. The provider reacts by fetching the content of the affected page from persistent store into a free page frame in real memory, updates the hierarchy of DAT tables respectively and then re-dispatches the consumer to resume execution with the now available data.

Benefits of memory virtualization

Virtual memory provides for economic use of real memory: It is important to realize that empty virtual memory pages do not need to be backed until they are accessed for the first time. This opens a huge potential for saving real memory, because very often consumers allocate virtual memory in large quantities, but with substantial subsets not being used for extended periods of time, or not even used at all. For example, the size of a virtual system might be defined as 4 GiB, but the guest operating system and its applications might use only a fraction of those 4 GiB.

The key here is the subtle distinction between *allocating* and *using* memory: Allocating or defining virtual memory establishes structures enabling the consumer to access virtual memory later-on. Only when the consumer actually uses (reads from/writes to) the virtual memory, it must be backed by real memory.

The lazy provisioning of real memory also provides for the possibility to allocate more virtual memory than real memory is available. This is called memory overcommitment and is discussed in Memory provisioning and memory overcommitment.

Virtual memory is pageable: A second opportunity for saving real memory opens when virtual memory is initially accessed at some point in time, but after that is accessed infrequently or not at all. This situation can be detected by the virtual memory provider that in turn may elect moving such low use pages to persistent store, thereby freeing corresponding real memory page frames for other use, such as backing a different virtual address space.

Virtual memory is sharable: If the virtual memory of several consumers exhibits identical regions, these regions can be shared among the consumers. Thus the same real memory can be used for backing the virtual memory of all these consumers, resulting in substantial savings of real memory. For example, when a Linux process forks, almost all virtual memory of the parent process is initially identical with that of the child process. Linux can share the real memory backing these identical parts. Only as virtual memory is modified by either process, these resultuing unique pages then require individual backing in real memory. This technique is known as copy on write.

Virtual memory is mappable: A fourth advantage of memory virtualization is its capability to map data into virtual address spaces. Two flavors are possible: Read only and read/write. The read only variant is mostly applied in order to map code elements such as Linux dynamic link libraries or z/VM shared segments into the virtual address space. This also provides for fast initialization of respective virtual memory regions, and is typically implemented along with sharing. The read/write variant could be used for fast file access, and when used along with sharing, provides a means of communication between virtual memory consumers.

Impacts of memory virtualization

Memory access latency: For the typical consumer of virtual memory, the process of providing virtual memory occurs transparently, except for the latency. In other words, the consumer of virtual memory does not notice that virtual instead of real memory is used, except that certain memory accesses are delayed until respective memory was made available by the provider. An example for such a delayed access might occur when hitting a virtual page not presently backed with a page frame in real memory.

This means that the main effect, which memory consumers observe when using virtual instead of real memory, is a delayed processing time of memory accesses, resulting in slower workload execution. It is of particular importance to realize that when real memory pressure exists, virtual memory consumers do **not** observe shortages of virtual memory. Particularly, the size of the virtual memory address space is unaffected, and does not depend on the availability of real memory in the first place. However, note that Linux provides controls that can limit virtual memory allocations depending on the availability of real memory For details see Memory provisioning and memory overcommitment.

Memory provisioning and memory overcommitment

In general, a virtual memory provider should make sure that the sum of the amount of real memory and the space available on paging disks is larger than the sum of the accessed pages of all virtual memory consumers.

The dilemma with that approach is that when initially providing a virtual address space of a certain size, a provider does not typically know in advance how much of that virtual memory will actually be used by the consumer. For example, the program executed by a Linux process might allocate 2 GiB of heap memory, but might only access a small fraction of that memory during later processing. Or a z/VM virtual system may be defined with 4 GiB, but only a fraction of that is actually used by the guest operating system.

For that reason, virtual memory providers may take the risk of allotting more virtual memory than they are able to back with real memory and paging disk space. However, if virtual memory usage at a later point in time requires more real resources than available, usually drastic measures result. For example, Linux might terminate selected processes in this case. The z/VM control program might issue a PGT004 abend when unable to obtain additional page slots on its paging devices.

In order to avoid such situations, system administrators must estimate the virtual memory usage of their workloads in advance, and assign sufficient real memory and paging disk space to their virtual memory providers.

Another approach available only under Linux is called the **overcommit handling mode**. Since kernel 2.5.30, the sysctl parameters vm.overcommit_memory and vm.overcommit_ratio can be used to customize the way Linux over-commits memory. Because in this paper the focus is on z/VM memory overcommitment, and our virtual systems are configured not to overcommit memory at the Linux process level, the use of these parameters is not further investigated here.

Interactions between virtual memory consumers and providers

In general, a direct interaction between the virtual memory consumer and the virtual memory provider is not required.

However, many virtualization platforms offer more or less sophisticated mechanisms for a direct interaction between virtual memory consumer and provider. When exploited, such mechanisms have the potential of greatly improving the overall performance of virtual systems and host system.

For example, the System z architecture provides facilities that enable the exchange of information about the state of virtual memory pages between the guest operating system and the z/VM control program (CP). Linux for System z exploits these mechanisms by means of the Collaborative Memory Management Assist (CMMA or cmm2) feature. For details, see *Linux on System z: Device Drivers, Features, and Commands* available at www.ibm.com/developerworks/linux/linux390/documentation_dev.html.

Another means of communication between CP and the guest operating system is known as page fault handshaking. When enabled (which is the default case), CP notifies the Linux on System z virtual system when a page fault is encountered. This enables Linux to select a different thread for dispatch, thereby using the time needed by CP to fetch the page from paging storage. Upon completion of the page read, CP sends another notification, enabling Linux to continue the previously interrupted thread.

For example, z/VM also provides the diagnose code x'10' (Release Pages) that enables a guest operating system to notify CP about the fact that a specified virtual memory range is no longer needed. As a result, CP can reclaim the real memory that had been backing the released virtual memory and use this freed real memory for other purposes. Note that releasing virtual memory in the described way does **not remove** that virtual memory from the virtual address space. Instead, it just communicates the virtual systems intent to not use the respective virtual memory range in near future.

Memory virtualization parameters

This topic offers background knowledge about which parameters and mechanisms are used to provide virtual memory in an efficient way to the memory consumers in the test environment.

Size of the dynamic paging area

A central parameter influencing the capability of z/VM to make memory available to virtual systems in an efficient way is the size of the dynamic paging area (DPA). The larger the DPA, the more page frames are available for backing virtual memory pages, reducing the need to write pages to a paging disk when page frames are needed for supporting the virtual memory of other virtual systems. z/VM dynamically determines the size of the DPA when initially loaded, by first reserving space for its own data structures in real memory, and then assigning the remaining real memory for the DPA. Thus the size of the DPA closely depends on the size of real memory available to z/VM. In our tests, we scaled down the amount of memory available to z/VM at IPL time, resulting in a smaller DPA and causing more pressure on the z/VM memory and paging subsystem in support of virtual system memory.

Memory overcommitment factor

A common definition of memory overcommitment is to add up the defined sizes of all virtual memory consumers, and relate that to the amount of real memory available:

$$(\sum \text{defined virtual memory of each virtual system}) : \text{z/VM real memory}$$

In this paper, we decided not to use this definition of memory overcommitment, because it does not eliminate the effects resulting from arbitrarily over-sizing virtual system memory.

Instead, we determined the memory overcommitment ratio as the ratio of instantiated memory related to resident memory:

$$ocr = \sum IMx : \sum RMx, \quad x=1...n$$

with:

`ocr` memory overcommitment ratio
`IMx` instantiated memory of virtual system x
`RMx` resident memory of virtual system x
`n` number of virtual systems considered

With this definition, a memory overcommit ratio of 1:1 means that z/VM system provides exactly the amount of memory that is in use by the considered set of virtual systems. A value larger than 1:1, such as for example 1.2:1, would mean that a shortage of real memory exists, or - in other words - that the amount of resident memory that z/VM maintains for the virtual systems is less than the amount of virtual memory in use by the virtual systems, that is, their aggregated instantiated memory.

When the memory overcommitment of *all* active virtual systems is to be considered in a memory constrained situation, the sum of the resident memory of all virtual system equals the z/VM DPA size. In this case, the above formula could be simplified to:

$$ocr = \sum IM_x : DPAsize, \quad x=1...n$$

with:

`DPAsize` z/VM DPA size (see the description of DPA size in z/VM performance metrics)

One challenge of this project was to determine how performance was impacted in memory constrained situations, namely when the amount of real memory supporting the virtual memory in use by the combined workload set was limited to such an extent that the workloads started to report errors. At this point the runs wi know

ere considered as failed, and not further included in the analysis.

Efficiency of memory management

The algorithms in use by operating systems for managing memory are very sophisticated. They have significant impact on the performance of processes working with virtual memory. With z/VM 6.3, memory management was completely redesigned. For details, see z/VM 6.3 CP Planning and Administration.

In our tests, we compared the performance of the combined workload set when executed under z/VM 6.3, as opposed to z/VM 6.2.

Efficiency of the paging subsystem

Another factor influencing virtual memory access performance is the efficiency of the paging subsystem. The more efficiently pages can be transferred between paging space on disk and the DPA, the faster virtual memory can be made available for use by virtual systems.

In our tests, we inspected the performance of running the combined workload set when using EDEV-SCSI based paging devices as opposed to using ECKD based paging devices. This comparison was of particular interest, because z/VM is able to overlap I/O operations on EDEV-SCSI based paging devices, but not on ECKD based paging devices.

System z processor and memory organization

In our tests, the z/VM system was operated within a logical partition (LPAR) defined on a System z central processor complex (CPC). For details, see Host system.

For System z CPCs, a packaging design based on so called *books* is used. A book is a hardware container that - among other elements - contains a multiple chip module (MCM) and memory. The MCM contains a number of chips, and each chip hosts several processor units (PUs), along with PU-specific caches as well as shared caches. The MCM further contains storage control (SC) chips that provide connectivity among PUs and to other books, particularly for access to memory. The SCs also host another level of caches that are shared by all PUs on the MCM.

This hierarchical architecture implies that memory is distributed across books, and caches are distributed across PUs, processor chips, and SCs. When memory is accessed across PUs, chips and books, different access times and latencies might result. We did not make efforts to detect or evaluate such situations as a source for performance variations.

Test results

The test results along with observations and conclusions are presented in this topic.

Comparison of different z/VM environments

This topic describes various further z/VM environments used to run the combined workload set.

The combined workload set was processed within different z/VM environments:

- A z/VM 6.3 installation using EDEV-SCSI based paging devices
  - With the database BI workload also using SCSI based devices
  - With the file system I/O workload using ECKD based devices
- A z/VM 6.3 installation using ECKD based paging devices
  - With the database BI workload using SCSI based devices
  - With the file system I/O workload also using ECKD based devices
- A z/VM 6.2 installation using EDEV-SCSI based paging devices
  - Except for the z/VM release, this environment was identical with the z/VM 6.3 EDEV-SCSI environment

Figure 3 shows the various z/VM configurations for each of the z/VM environments that were used for running the combined workload set.

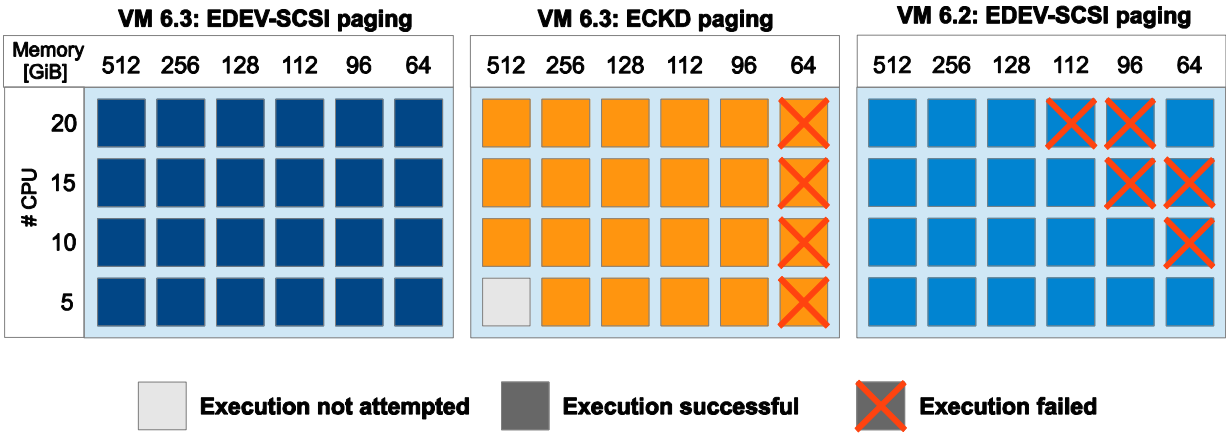

Figure 3: Executions of the combined workload set within different z/VM environments

Observations:
Some attempted test executions failed. A failure was assumed when any of the workloads did not successfully perform its complete test execution.

The reasons for failures were different:

- In the case of the z/VM ECKD paging environment, all executions for the 64 GiB z/VM real memory case failed, because the high level variant of the page-cached file system I/O workload did not complete in time. A potential reason for the failure might have been that the file system I/O workload also used ECKD devices on the same storage system and received error conditions on these devices while z/VM produced high paging rates.
- In the case of the z/VM 6.2 EDEV-SCSI paging environment, the transactional WAS workload encountered so many HTTP communication failures that an internal threshold was exceeded and caused the workload termination. In some cases, the database BI workload also encountered communication errors causing the workload to fail.

- In case of workload failures we do not further consider the test execution results and other metrics from z/VM and the other workloads because the failure of one workload left z/VM resources vacated for use by other workloads, making the comparison with successful combined workload set executions impossible.

Conclusions:

- **Environment z/VM 6.3: EDEV-SCSI paging**
  In this environment, z/VM paging, as well as the database BI workload, shared parts of the I/O subsystem and the storage server. In spite of this resource overlap we did not observe strong degradations of either the z/VM paging bandwidth or of the database BI workload I/O activity. In this environment we did not observe any complete workload failure, albeit increasing amounts of recoverable communication errors were observed for the transactional WAS workload when z/VM real memory was more constrained.
- **Environment z/VM 6.3: ECKD paging**
  In this environment, z/VM paging, as well as the file system I/O workload, shared parts of the I/O subsystem and the storage server. This resource overlap caused the high level variant of the page-cached file system I/O workload to fail in the most constrained 64 GiB z/VM real memory configurations. Furthermore, in less memory constrained configurations where the file system I/O workload did not fail, the partial sharing of I/O resources between z/VM paging and other I/O activities might have accounted for z/VM being retarded from achieving its full page I/O bandwidth.
- **Environment z/VM 6.2: EDEV-SCSI paging:**
  Most of the failure situations we observed occurred in the z/VM 6.2 environment. Apparently in that environment already a moderate overcommitment of z/VM real memory and real CPUs caused more severe constraints within the workload virtual systems, resulting earlier in workloads failures.

Analysis of overall workload results

Figure 4 shows the relative throughput of the overall workload for the various z/VM environments that were analyzed while the combined workload set was executed in various z/VM configurations.

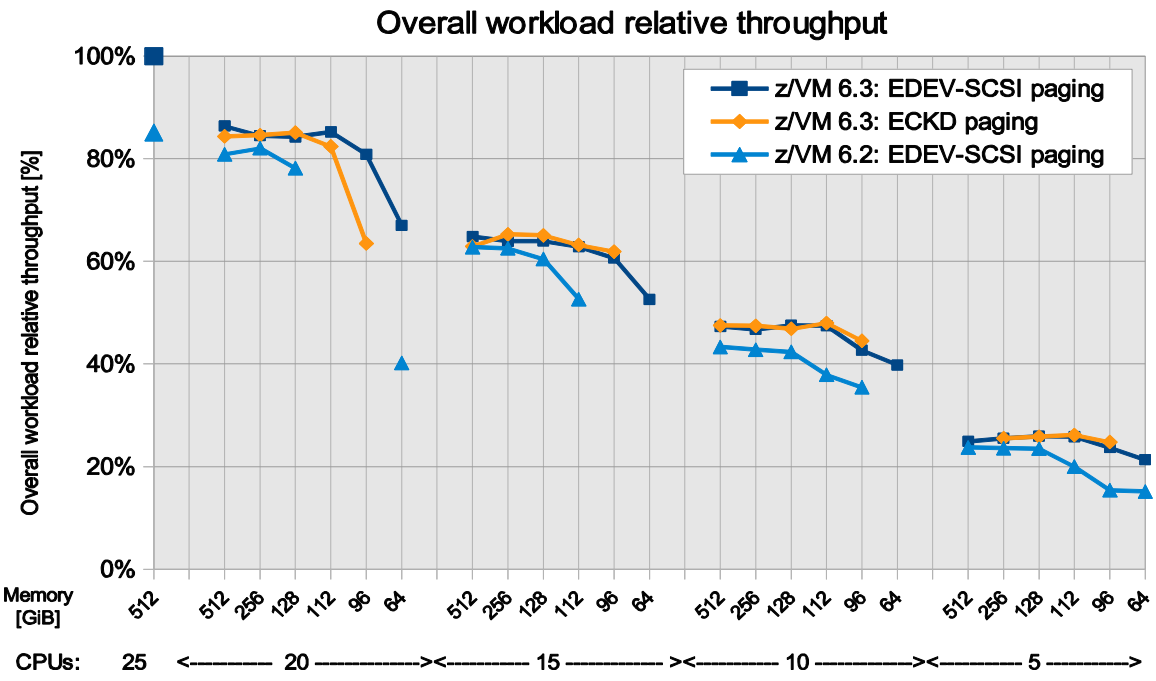

Figure 4: Overall workload relative throughput

Note: The results of the *z/VM 6.3: ECKD paging* environment are relative with respect to the reference execution of the z/VM 6.3: EDEV-SCSI paging environment, because in the absence of z/VM paging activity these two environments were assumed to behave identically.

Observations:
When 15 or less real CPU were configured for z/VM 6.3, the overall workload relative throughput exhibited similar values for the environment using EDEV-SCSI paging devices and for the environment using ECKD based paging devices. However, when 20 real CPU were configured for z/VM, the overall throughput degraded faster for the ECKD configuration than for the EDEV-SCSI configuration. Furthermore, when 64 GiB z/VM real memory were configured in the z/VM 6.3: ECKD paging case, all test executions failed. Likewise, some test executions in the z/VM 6.2: EDEV-SCSI environment failed. For details, see Figure 3.

The z/VM 6.2 environment overall produced lower throughput values. Furthermore, when z/VM resources were not constrained (CID 512/25 on the far left side in Figure 4), the overall throughput was lower than that achieved in the unconstrained z/VM environment using EDEV-SCSI paging.

Conclusions:
First of all, note that the overall workload relative throughput is a rather coarse metric, because it treats all workload variants with the same weight, leveling the rather unique characteristics of the workloads. That notwithstanding, the results presented in Figure 4 unfold a surprisingly consistent view.

The z/VM 6.3 environment using EDEV-SCSI paging consistently exhibited very high overall throughput values. Even in the most memory constrained configurations with only 64 GiB real memory, no workload failures occurred. This makes this environment all in all the best performing and most reliable one for processing the combined workload set.

With only moderate z/VM real memory constraints, the z/VM 6.3 environment using ECKD paging devices performed similarly to the one using EDEV-SCSI paging devices. However, the failures occurring when z/VM was configured with 64 GiB real memory, and the earlier throughput drop when z/VM was configured with 20 real CPUs were weak points for this environment.

The z/VM 6.2 environment using EDEV-SCSI paging consistently exhibited comparatively low overall throughput values. A factor might have been that the traditional z/VM scheduler and the memory subsystem in use in the z/VM 6.2 environment were less capable than their completely redesigned z/VM 6.3 counterparts.

Analysis of z/VM paging

Figure 5 shows the average page read rates from the paging devices for the various z/VM environments that were analyzed while processing the combined workload set in various z/VM configurations.

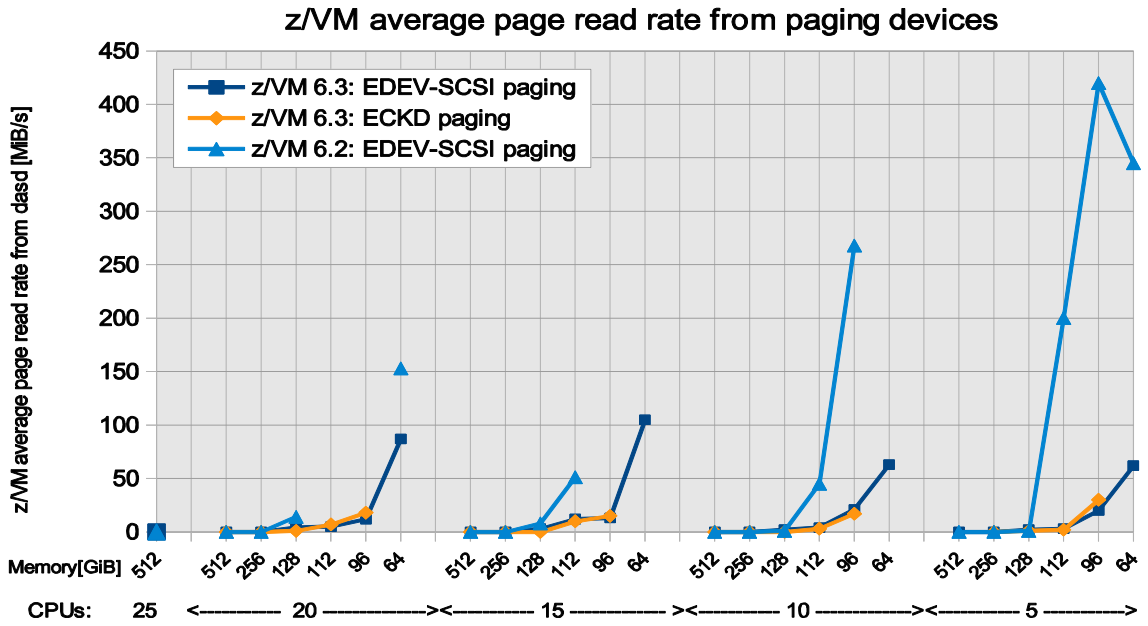

Figure 5: z/VM average page read rates from paging devices

Observations:
Within the z/VM 6.3 environments, we observed average page read rates that progressively increased as z/VM real memory became more constrained. Within the z/VM 6.2 environment, the increase of the page read rates was much steeper than within any z/VM 6.3 environment, and reached very high values. Again, missing data points indicate a failing test execution, see Figure 3.

Conclusions:
Except for the z/VM 6.2 environment, the growth of the z/VM average page read rate seems reasonable for addressing the increased pressure on z/VM real memory. Also in these cases, the paging I/O rates (not shown in Figure 5) were well within the capabilities of the paging devices.

As explained in the workload analysis, the overall workload relative throughput values obtained in the z/VM 6.2 environment were below those achieved in the z/VM 6.3 environment. Thus, the very high average page read rates observed in the z/VM 6.2 environment did not result in higher virtual memory availability. In addition, the use of expanded storage (as recommended for z/VM 6.2) did not help to attenuate the z/VM real memory constraints to the extent achieved by z/VM 6.3 environments. Apparently, the traditional paging subsystem of the z/VM 6.2 environment already reached its limits when moderate z/VM real memory constraints existed.

Analysis of z/VM CPU load

Figure 6 shows the z/VM user CPU load and the CP attributed to guest CPU load, summed up for all virtual systems. Recall that the CP attributed to guest CPU load is part of the user CPU load.

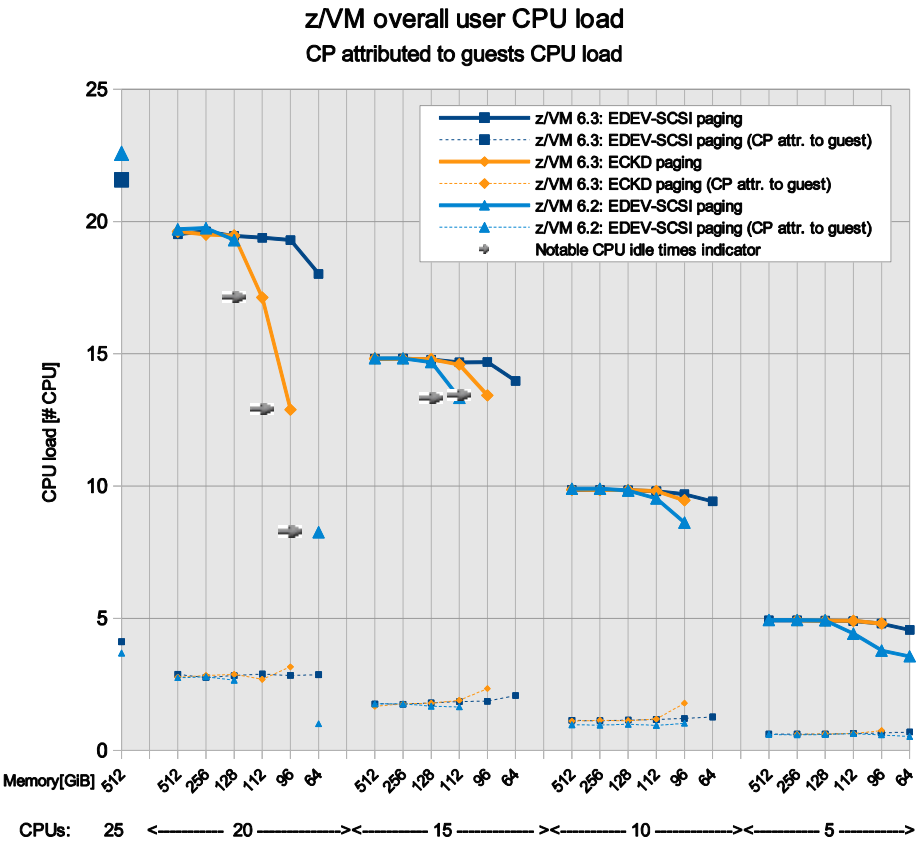

Figure 6: z/VM overall user CPU load and CP attributed to guest CPU load

Observations:
As stated before, missing data points indicate a failing test execution, see Figure 3.

When 128 GiB or more real memory was configured for z/VM, the combined user CPU load stayed very close to the amount of real CPUs configured for z/VM. With 96 or less real memory configured for z/VM, the CPU load values diverged for the different z/VM environments.

When higher numbers of real CPU were configured for z/VM, the effect of the reduction of z/VM real memory on the user CPU load was more significant. In some cases (marked with a gray arrow in Figure 6), notable real CPU idle states (real CPU wait times) were observed (0.5 CPUs, or more). Only when real CPU idle times do not occur, a reduction of the user CPU load directly corresponds to an increase of the system CPU load.

On the other hand, when - in a CPU constrained situation - real CPU idle times occur, these indicate the fraction of real CPUs that are waiting for prerequisite activities - such as a page-in operations - to complete before being able to process work pending on virtual CPUs. In other words, in a CPU constrained situation real CPU idle times indicate that there is excess CPU power available in the environment that cannot be used because other parts of the environment (such as real memory) are over-committed.

The z/VM 6.3 environments were least affected by z/VM real memory constraints, when the number of z/VM real CPU was kept constant. At the same time, the CP attributed to guest CPU load was almost unaffected by z/VM real memory constraints, except for the z/VM 6.2 environment when 20 real CPUs were configured for z/VM.

Figure 7 shows the z/VM system CPU load.

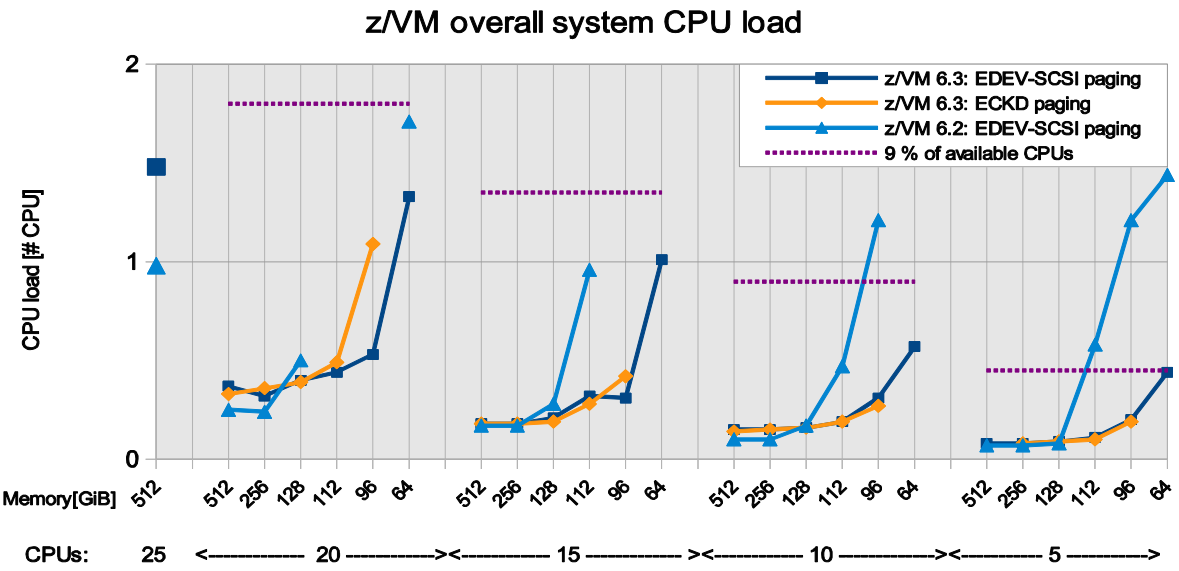

Figure 7: z/VM overall user CPU load and CP attributed to guest CPU load

Observations:
The z/VM overall system CPU load progressively increased, when the available z/VM real memory was constrained. Except for the z/VM 6.2 environment, the overall system CPU load stayed below 9 % of the real CPU capacity. The system CPU load behaved very similar to the page-read rates (see Figure 5). Again, missing data points indicate a failing test execution, see Figure 3.

Conclusions for user and system CPU load:
For paging subsystems using EDEV-SCSI based paging devices, an increase of the system CPU load is to be expected, because SCSI I/O implies real CPU activity for data transfer operations. Opposed to that, paging subsystems using ECKD based devices are relieved of many I/O related activities through the use of system assist processors (SAPs). Nevertheless, also for the z/VM 6.3 environments using ECKD based paging devices, we observed an increase of the system CPU load, as z/VM real memory became more constrained.

A significant portion of the system CPU load is assumed to be caused by the higher paging activity caused by the lack of z/VM real memory. Especially in the z/VM environments with only 64 GiB real memory - which is less than half the real memory used during the reference execution - the virtual memory access latency caused by the lack of real memory and resulting z/VM paging activity is a key parameter for the overall performance. The faster the paging subsystem is the better the whole system is performing. Workloads with a memory use pattern that causes larger fractions of their virtual memory being kept resident by z/VM have an advantage in this strongly memory constrained situation. Likewise, workloads are advantaged that are able to process alternate execution paths when being notified by z/VM about a page fault (see Interactions between virtual memory consumers and providers.

Overall, the system overhead resulting from constraints of z/VM real memory and/or real CPUs was very well within reasonable limits, except for the z/VM 6.2 environment. Even in the most memory constrained z/VM 6.3 configurations, more than 94 % of the available real CPU capacity was used for user workload processing.

Workload analysis
This topic presents the workload specific results of the study. Results from each individual workload from the combined workload set is presented in a separate subtopic. Because of the amount of data, detailed results are exposed only for the z/VM 6.3 environment with EDEV-SCSI paging. This environment is well suited to uncover the basic interrelations.

Reference result describes the reference workload result, which sets the base against which all other measurements are compared.

In Scaling memory and number of processors, the overall z/VM CPU load behavior is inspected when running the combined workload set in selected resource constrained z/VM configurations.

Finally, the test results are described that resulted from executing each individual workload from the combined workload set in z/VM configurations, where the amount of real memory or the number of real CPUs, or both, were reduced. For details, see Test configurations and Test procedure.

Each workload is described in a separate topic. The focus in these topics is on analyzing the individual workloads with respect to their throughput, generated CPU load, memory usage and generated paging workload.

Reference result

Running the combined workload set in an unconstrained z/VM environment was called the reference execution. Unconstrained means that an abundance of z/VM real memory was available such that no z/VM paging occurs, and that an abundance of z/VM real CPUs was available such that not all of these CPUs were loaded to their limit. We determined that 512 GiB real memory and 25 real CPUs satisfied these prerequisites.

The reference result is the result of the reference execution.

Table 17 lists the average CPU load and the average memory consumption for the individual workloads that was observed when performing the reference execution.

Table 17. Reference result: Average CPU and memory consumption of the combined workload set

#	Workload	Level	Component or variant	CPU [# proc]		Memory [MiB]	
				defined	CPU load	defined	instantiated
1	Database BI			4	2.82	307,200	83,968
2	Transactional WAS	medium	IHS	1	0.58	700	560
3			WAS	2	1.18	4,096	2,232
4			DB2	1	0.92	2,048	2,048
5		high	IHS	1	0.86	700	575
6			WAS	4	1.72	4,096	2,222
7			DB2	2	1.27	2,048	2,048
8	File system	medium	page cached	2	0.92	16,384	12,288
9			direct I/O	2	0.88	16,384	2,543
10		high	page cached	4	1.60	16,384	16,384
11			direct I/O	4	1.22	16,384	2,572
12	Java	medium		1	0.92	4,096	3,626
13		high		2	1.82	4,096	3,579
14	Network	medium	client	1	0.79	1,024	467
15			server	1	0.94	1,024	436
16		high	client	2	1.49	1,024	465
17			server	2	1.65	1,024	443
Sum:				36	21.57	398,712	136,456

Observations:
No individual workload (or workload component) used all of its assigned CPU power in terms of virtual processors and processor share.

Likewise, most of the workloads did not make use of all of their virtual memory. However, some workload components, such as DB2 and the high level file system workload using the page cache, made use of all their virtual memory.

Apparently the workload level has almost no impact on the instantiated memory of the guests, except for the file system I/O workload with page cache. This is expected because here the variation in the workload level is related with a change in the amount of files in use.

The aggregated amount of z/VM processing power used by the combined workload set running in an unconstrained environment was about 21.6 CPUs, and the aggregated amount of required real memory - which in this unconstrained environment is identical with the aggregated amount of instantiated memory - was about 133 GiB (1 GiB = 1024 MiB).

Conclusions:
The amounts of real resources needed are significantly smaller than the accumulated amounts of virtual resources configured for the virtual systems. This confirms the statement made earlier that the more or less arbitrarily configured sizes are not a good basis for determining resource overcommitment (see Memory overcommitment factor). Instead, in this paper we define resource overcommitment with respect to the reference result.

Scaling memory and number of processors

In this section, the test results are inspected for selected z/VM configurations. The focus of the results is on overall z/VM CPU load and on z/VM paging behavior.

For z/VM configuration details, see Test configurations. For the test procedure, see Test procedure.

z/VM CPU overcommitment

Figure 8 shows a breakup of the z/VM CPU loads observed when the combined workload set was executed within several constrained z/VM configurations.

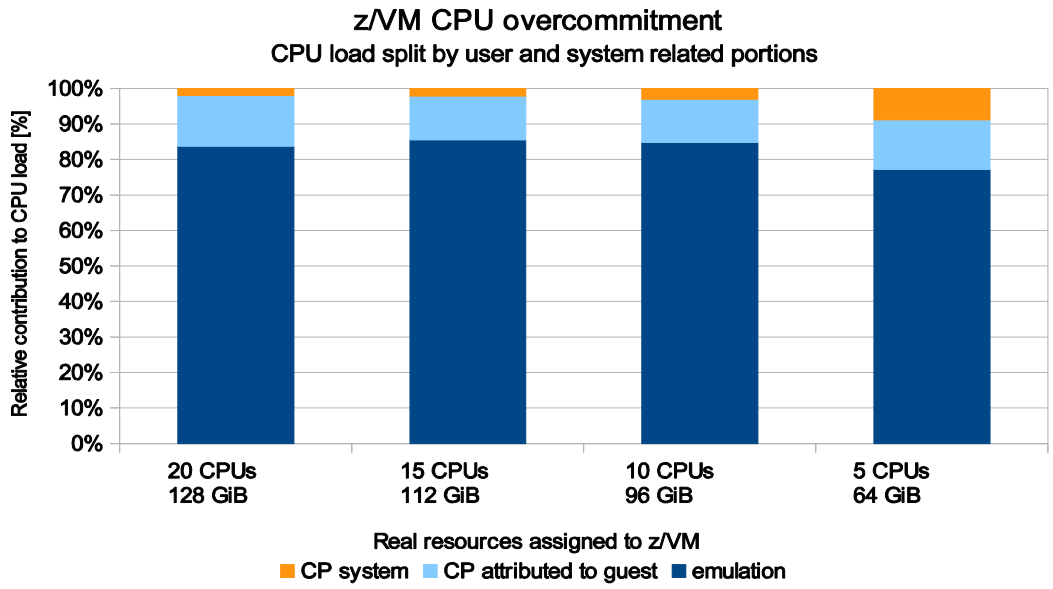

Figure 8: Breakup of z/VM CPU usage when real CPU and real memory are constrained

In order to emphasize the distribution of CPU loads for each z/VM configuration, in Figure 8, all results are scaled, such that all columns extend to 100 %.

Observations:
The fractions of CP system CPU load, emulation CPU load, and CP attributed to guest CPU load stay rather constant while the number of real CPUs is reduced to 10, and the amount of real memory is reduced to 96 GiB. Below that, the fraction of the CP system CPU load starts to rise.

Conclusions:
As long as real resource constraints stayed within tolerable limits, z/VM is quite well capable to process virtual system workload without increased system expenses. If resources are further constrained, apparently additional system expenses result. For example, we observed that system paging activity rises significantly as real memory is limited to values below 96 GiB. Recall that during the reference execution, the combined instantiated memory, that is, the sum of the virtual memory actual in use by all virtual systems, was found to be 133 GiB. Thus, when real memory was 96 GiB or below, significant z/VM real memory pressure did result. Nevertheless, the used z/VM 6.3 test environment remained capable to process the workload even in that situation, albeit with performance degradations, but without workload failures.

Database BI workload

This section describes the analysis of the test results from processing the database BI workload under various constrained z/VM configurations.

Workload analysis
Figure 9 shows the relative throughput of the Database BI workload when processed as part of the combined workload set within a variety of differently constrained z/VM configurations.

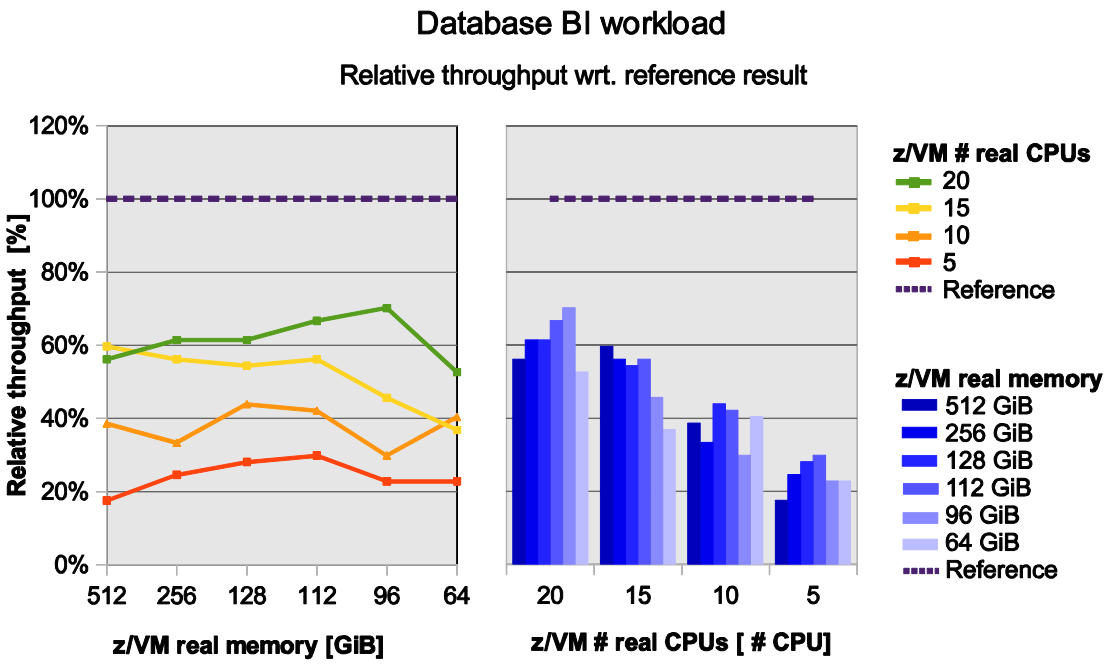

Figure 9: Relative throughput of the Database BI workload

General description of relative throughput diagrams
Figure 9 (and similar figures in subsequent test results) depict the relative throughput of the workload with respect to the reference result.

The left side of Figure 9 emphasizes the dependency of the workload relative throughput on the z/VM real memory configuration. It depicts the average relative throughput for this workload over the amount of real memory configured for z/VM, parametrized by the amount of real CPUs configured for z/VM.

The right side of Figure 9 emphasizes the dependency of the workload's relative throughput on the z/VM real CPU configuration. It shows the same set of values as the left side, but this time grouped by the amount of real CPUs configured for z/VM, and within these grouped by the amount of real memory configured for z/VM.

In both cases, the dark dotted line indicates the 100 % throughput achieved during the reference execution.

Observations:
The right side of Figure 9 shows that a corresponding reduction of the throughput results, when less real CPUs were configured for z/VM. Depending on the number of real CPUs configured for z/VM, this general behavior is more or less interfered by the amount of real memory configured for z/VM.

From the left side of Figure 9, it appears that with the z/VM configurations using 20 or 5 real CPUs, the workload throughput was comparatively low when z/VM was configured with 512 GiB real memory, but then gradually increased as z/VM was configured with less real memory. Only after the amount of z/VM real memory went below a certain limit, the relative throughput decreased again.

The throughput that came most closely to that achieved during the reference execution was 70 %, using a z/VM configuration with 96 GiB real memory and 20 real CPUs.

With the z/VM configurations using 15 real CPUs, the throughput gradually decreased as less z/VM real memory was configured. It is noteworthy that when 512 GiB real memory were configured for z/VM, the throughput was 60 % - slightly higher than with 20 real CPUs.

The z/VM configurations using 10 real CPUs produced throughput results that fluctuated between 30 % and 44 %.

Conclusions:
The Database BI workload seems to be more impacted by z/VM real CPU constraints as opposed to z/VM real memory constraints. Furthermore, the influence of the amount of real memory configured for z/VM varies depending on how many real CPUs are configured.

When either 20 or 5 real CPUs were configured for z/VM, the Database BI workload was able to reclaim throughput in spite of the fact that the amount of real memory configured for z/VM was decreased. Apparently in these cases the other workloads suffered more than the Database BI workload from the z/VM real memory constraints, preventing them to make full use of other resources such as CPU and I/O bandwidth. The Database BI workload in turn could apparently claim these resources, resulting in a relative throughput increase.

This is a very important aspect to be always considered when evaluating the results from this study. It shows not only the effects resulting from real resource shortage, but also shows that the results were additionally influenced by the behavior of the other workloads.

The behavior of the Database BI workload, when processed in real memory constrained z/VM configurations, exhibited an increasing throughput when z/VM real memory was reduced. This is very untypical when compared the throughput of the most other workloads, which stay constant and then decrease. One probable reason for this behavior might be that databases are very sophisticated systems that might be able to compensate the impacts resulting from memory access latencies by efficiently caching data in buffer pools, and by asynchronously scheduling operations.

CPU usage analysis shows that indeed in the 20 real CPU case, the Database BI workload for most z/VM real memory configurations was able to consume more CPU than during the reference execution.

CPU usage analysis

Figure 10 shows the average CPU load for the Database BI workload when processed as part of the combined workload set within a variety of differently constrained z/VM configurations.

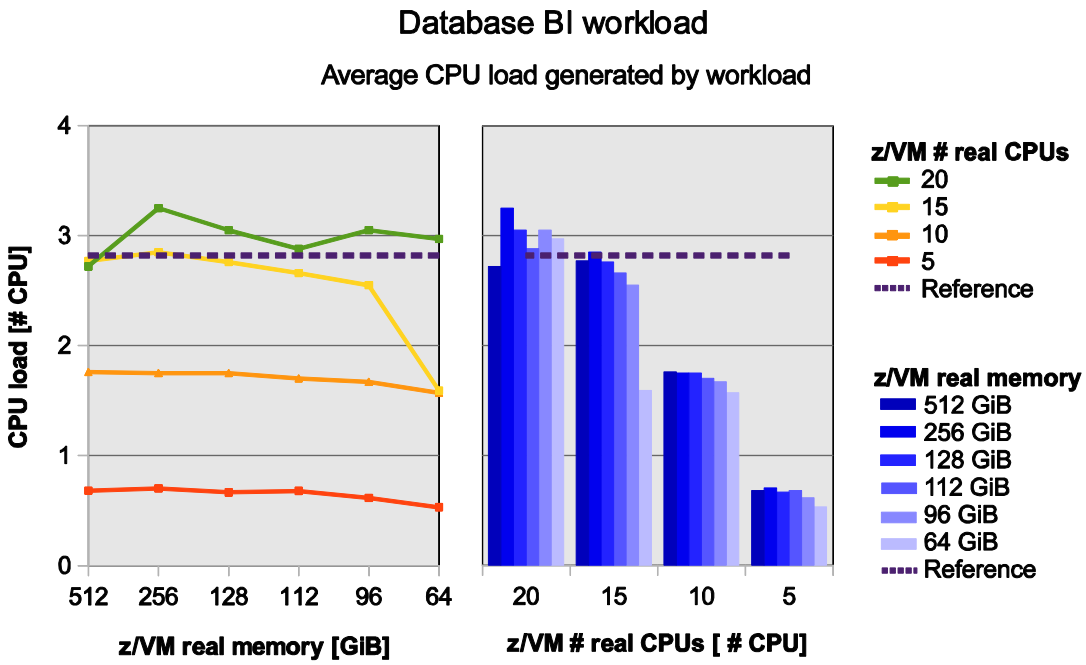

Figure 10: CPU load generated by workload

A general description of the elements shown in Figure 10 is given in *General description of relative throughput diagrams* (page 35), except that in Figure 10 CPU load is presented instead of throughput.

Observations:
When 20 real CPUs were available to z/VM, the Database BI workload for all except the 512 GiB z/VM memory setting, generated a higher average CPU load at a lower throughput than during the reference execution (see Figure 9).

When 15 real CPUs were available to z/VM, the average CPU load decreased slowly when z/VM was configured with 112 GiB real memory. The decrease was more pronounced when 96 GiB or less of real memory were configured for z/VM.

When 10 or less real CPUs were configured for z/VM, the dependency of the average CPU load on the amount of real memory became less significant, and there was almost not dependency when only 5 real CPUs were configured for z/VM.

Conclusions:
The CPU load behavior of the Database BI workload reacts more strongly on CPU constraints, as opposed to constraints on the amount of real memory. For example, halving the amount of real memory configured for z/VM from 128 GiB to 64 GiB when z/VM was configured with 10 or 5 CPUs, caused only slight reductions of the average CPU load. By contrast, when halving the amount of real CPUs from 20 to 10 with all but the lowest z/VM real memory setting, the average CPU load was almost halved as well.

When 20 real CPUs were configured for z/VM, the average CPU load generated by the Database BI workload was even higher than that generated during the reference execution. In other words, when running in a memory constrained z/VM environment, but with sufficient processing power available to z/VM, the Database BI workload reclaimed processing power, apparently at the expense of other workloads. This was already suspected when analyzing the workload performance, see the workload analysis. However, this gain of CPU power did not result in an absolute increase of workload throughput. Apparently, in these cases, the Database BI workload lacked other resources required for its workload processing.

Memory usage and paging workload analysis for CID 64/20

A detailed memory usage and workload analysis is only presented for the z/VM configuration with 64 GiB real memory and 20 real CPUs (CID 64/20). We consider this configuration as most suited to outline the effects resulting from z/VM real memory and real CPU constraints.

Figure 11 shows the average page distribution and the average page read rates for the Database BI workload when processed as part of the combined workload set when z/VM is running with the CID 64/20 configuration.

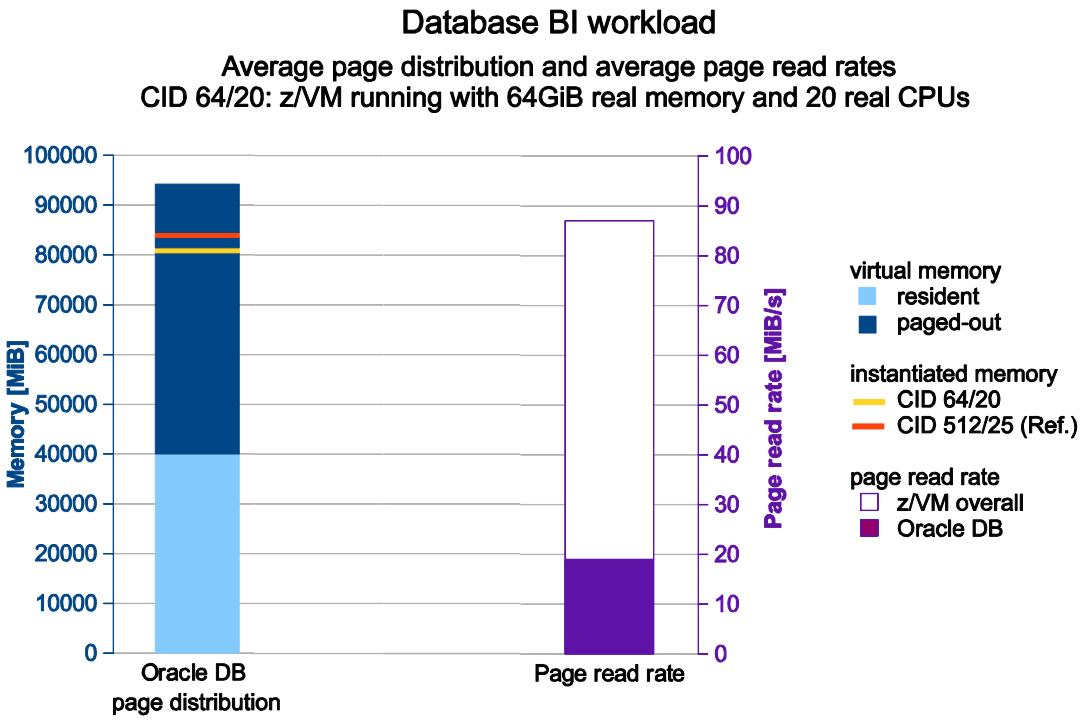

Figure 11: Database BI workload within CID 64/20: Average page distribution and average page read rates

General description of page distribution and page read rate diagrams

In Figure 11 (and similar figures in subsequent test results), the blue columns on the left side depict the page distribution for the virtual system running the workload or workload component.

- The lower part (bright blue) represents the virtual system's resident memory.
- The upper part (dark blue) represents the stored memory.
- The bright yellow line marks the virtual system's instantiated memory. For comparison, the bright red line shows the virtual system's instantiated memory during the reference execution.

Note: When similar diagrams are shown elsewhere in this document, different memory scales may be applied.

The magenta column on the right side depicts the page read rate for the virtual systems running the workload. If the workload is comprised of more than one component, the page read rate of each component is shown in a different shade of magenta. For comparison, the page read rate for the whole z/VM system is also depicted (hollow magenta).

Observations:
On average, the Database BI workload used about 62 % (39936 MiB of 64783 MiB) of the z/VM DPA. The average instantiated memory of 80896 MiB was about 3.7 % less than that of the reference result (83968 MiB). About half of the instantiated memory was kept resident by z/VM, while the other half was stored on paging devices.

Note that the sum of resident memory and stored memory is larger than instantiated memory, indicating that a certain amount of virtual memory is backed on both locations: in z/VM real memory and on z/VM paging devices.

Also note that the instantiated memory within CID 64/20 is less than that observed during the reference execution.

On average, the Database BI workload accounted for about 22 % (20 MiB/s from 88 MiB/s) of the z/VM page read operations. About 0.05 % (19.414 MiB/s / 39936 MiB) of the virtual system's resident memory were in the process of being paged in per second. At the same time - not shown in Figure 11 - about another 0.05 % (19.628 MiB/s / 39936 MiB) were in the process of being paged out per second.

Conclusions:
The Database BI workload is by far the largest consumer of virtual memory from the combined workload set - resulting in the use of almost two thirds of the z/VM DPA size. With that large amount of virtual memory kept resident, the Database BI workload consumed only about a quarter of the bandwidth of the z/VM paging subsystem.

For the Database BI workload, the dominant parameters determining the virtual memory consumption (beyond the memory usage imposed by the Linux system itself) are the Oracle system global area (SGA, configured 200 GiB) and the program global area (PGA, configured 45 GiB) sizes.

A rather low number of workload users (16) was used to drive the Database BI workload in order to keep the workload CPU usage within practical limits. As a consequence,

- The amount of allocated shared memory was 67 GiB (as reported by Linux sar report). The shared memory usage is a metric for the actually used fraction of the SGA. In this case, it is still larger than the average amount of resident memory. In other words, at least a fraction of the shared virtual memory is backed as stored memory, that is, it is stored on z/VM paging devices.
- The amount of memory really used for the PGA is about 100 MiB (according to the Oracle AWR report).
- The memory requirement for Linux page tables was small (about 60 MiB), even though no large pages were used for the Oracle memory pools.

Apparently, the most important pages for the database could be kept in real memory. This seems to make the Database BI workload well suitable for memory over-committed environments. This assessment is even strengthened by the fact that the instantiated memory for the database virtual system is about five times larger than for the other largest guest (configured with 16 GiB virtual memory), and about twenty times larger than that of a typical 4 GiB guest.

Recall that if upon virtual memory access a page needs to be read from the z/VM paging devices, a significant performance impact results, because of the much lower disk access speed compared to the real memory access speed. Thus, a high page read rate is a first indicator of a potential performance impact. Considering the large guest size used for the database (300 GiB), the observed page read rate of 20 MiB/s still seems moderate. Apparently, the database is able to hide the impact on total throughput, for example by asynchronously scheduling operations.

Transactional WAS workload

This section describes the analysis of the test results from processing the transactional WAS workload under various constrained z/VM configurations.

This workload was executed on a medium and on a high level.

Workload analysis

Figure 12 shows the relative throughput of the medium level transactional WAS workload when running as part of the combined workload set within a variety of differently constrained z/VM configurations.

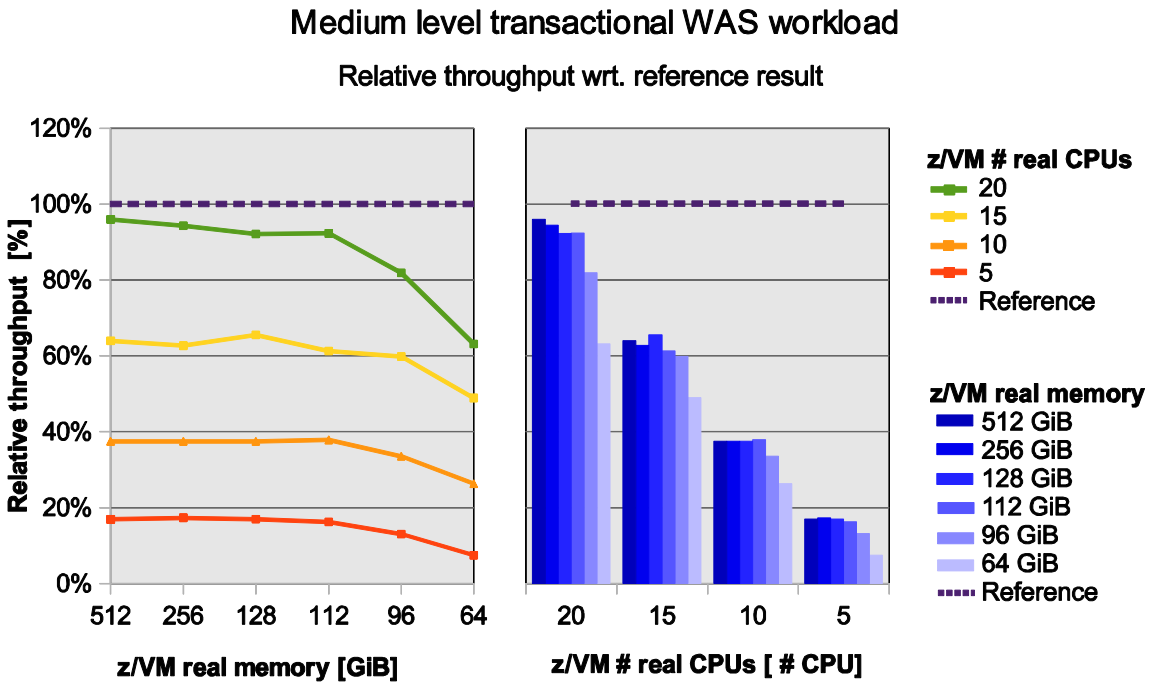

Figure 12: Medium level transactional WAS workload: Relative throughput

Figure 13 shows the relative throughput of the high level transactional WAS workload.

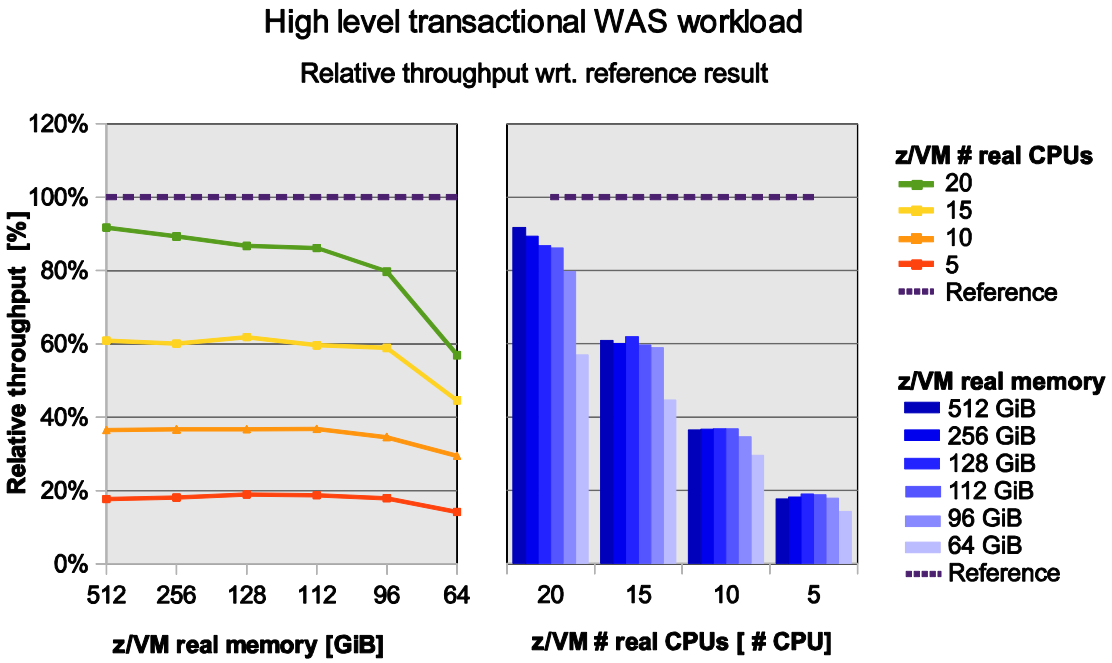

Figure 13: High level transactional WAS workload: Relative throughput

A general description of the elements shown in Figure 12 and Figure 13 is given in *General description of relative throughput diagrams* (page 35).

Observations:
Figure 12 shows that the degradation of the medium level Transactional WAS workload relative throughput stayed relatively moderate until less than 112 GiB real memory were configured for z/VM. Particularly, with 64 GiB real memory, the throughput decline is more significant.

The tendency to remain unaffected by constraints of z/VM real memory was more pronounced for the throughput of the high level variant shown in Figure 13, particularly when small numbers of real CPUs were configured for z/VM.

By contrast, the influence of constraining the amount of real CPUs configured for z/VM while the amount of real memory configured for z/VM was kept constant, was much more significant at both workload levels. When the configured z/VM real memory was below 96 GiB, the relative throughput did not decline as sharply than above 96 GiB.

Conclusions:
The transactional WAS workload reacted more sensitively on z/VM CPU constraints as opposed to z/VM real memory constraints. It exhibited a nearly linear dependency of the throughput on the number of real CPUs configured for z/VM.

Opposed to that, the transactional WAS workload could tolerate even significant reductions of the amount of real memory configured for z/VM, with only small throughput degradations. This behavior was even more articulated for the high level transactional WAS workload, particularly when 15 or less real CPUs were configured for z/VM, and as long as the amount of real memory configured for z/VM stayed above 64 GiB. The throughput stayed almost constant until a certain level of memory overcommitment was reached, but beyond that level, the

performance impact became severe. A possible explanation could be that in a z/VM real memory constrained situation, substantial portions of the Java heap cannot be kept constantly resident. When access to those portions is needed - for example during a Java garbage collection - z/VM needs to page in respective virtual memory regions, causing memory access latencies for the virtual system.

The `gencon` garbage collection policy that is per default applied by WebSphere 8.5, might have attenuated these effects as it tends to less frequently perform passes over the complete Java heap. We will see later during the analysis of the Java workload, that use of the `optthruput` garbage collection policy might impose higher memory usage.

Overall, the transactional WAS workload seems to be a good candidate to be operated in conditions where z/VM real memory constraints exist, as long as the z/VM installation has a healthy paging subsystem available (see also section *Memory usage and paging workload analysis for CID 64/20*).

CPU usage analysis

Figure 14 shows the average CPU load for the medium level transactional WAS workload added up from all three workload components (IHS, WAS, and DB2), and when running as part of the combined workload set within a variety of differently constrained z/VM configurations.

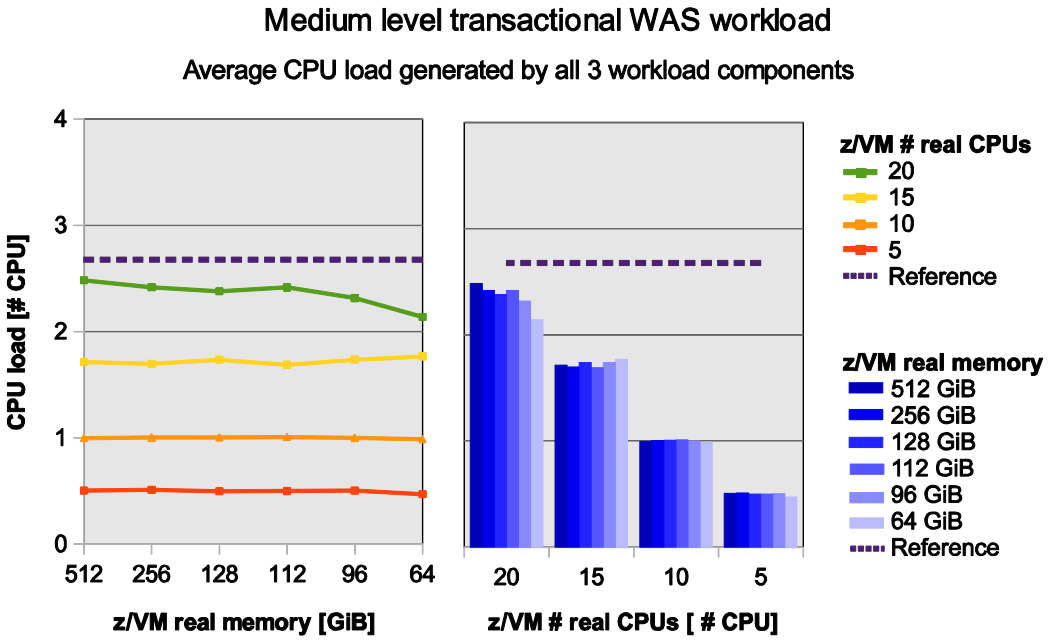

Figure 14: Medium level transactional WAS workload: A verage CPU load generated by all three workload components

Figure 15 shows the average CPU load for the high level transactional WAS workload added up from all three workload components.

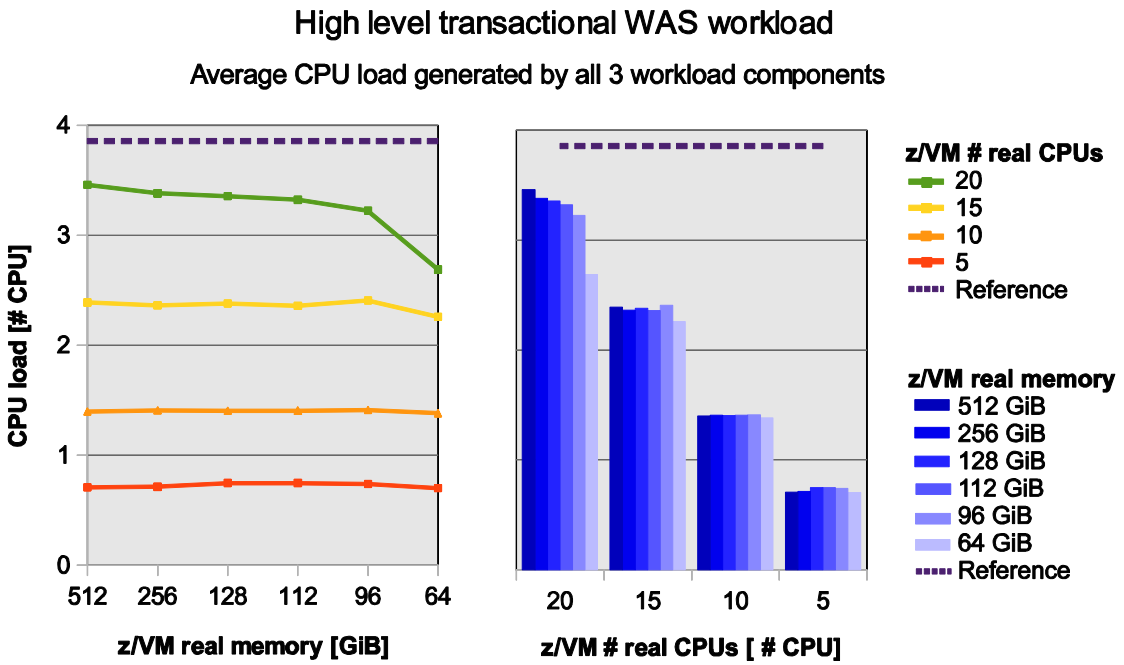

Figure 15: High level transactional WAS workload: Average CPU load generated by all three workload components

A general description of the elements shown in Figure 14 and Figure 15 is given in *General description of relative throughput diagrams* (page 35), except that in Figure 14 and Figure 15 CPU load is presented instead of throughput.

Observations:
For both levels of the transactional WAS workload, the generated CPU load is almost independent from the amount of real memory configured for z/VM. Only when 20 CPUs were configured for z/VM, the CPU load generated by the transactional WAS workload decreased noticeably when 96 GiB or less were configured for z/VM.

On the other hand, when the number of CPUs configured for z/VM was constrained, the CPU load generated by the transactional WAS workload decreased nearly linearly with the number of CPUs configured for z/VM.

Conclusions:
As stated already in the workload analysis, the transactional WAS workload apparently is CPU bound.

A closer look into the CPU load behavior of the various components of the transactional WAS workload shows possible deviations of individual workload components.

CPU load distribution

Figure 16 shows the CPU load generated by the three components of the medium level transactional WAS workload.

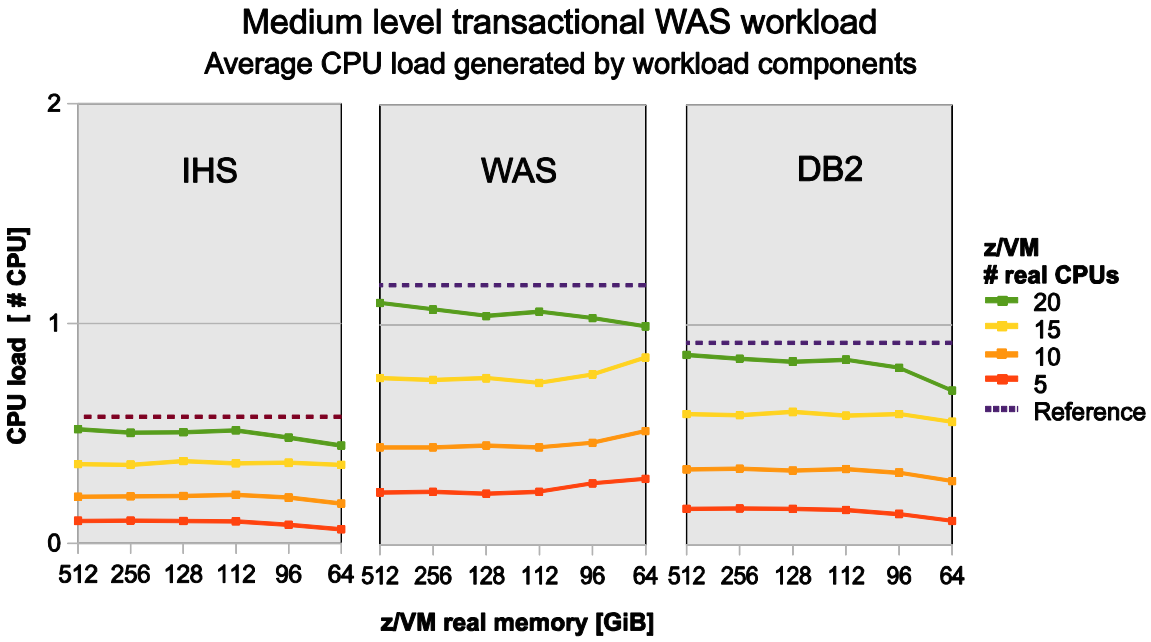

Figure 16: Medium level transactional WAS workload: CPU load generated by workload components

Figure 17 shows the CPU load distribution generated by the three components of the high level transactional WAS workload.

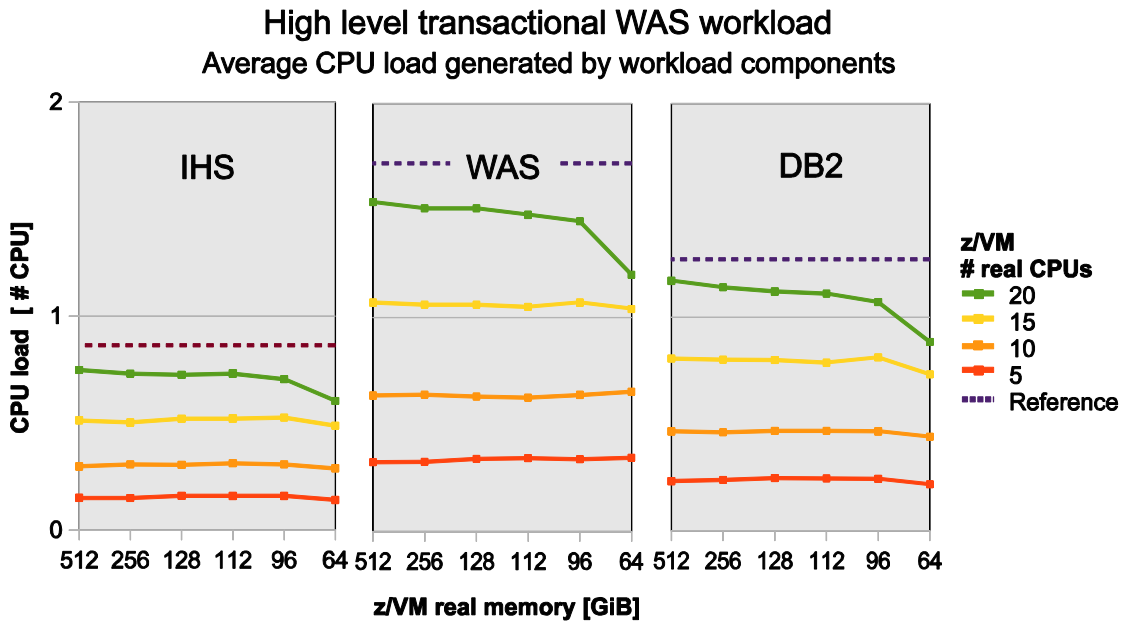

Figure 17: High level transactional WAS workload: CPU load generated by the workload components

Observations:
The three components of the transactional WAS workload exhibited similar CPU load characteristics. A significant impact of z/VM real memory constraints on the CPU load of the various components of the transactional WAS workload was only observed when 20 real CPUs were configured for z/VM, and was more pronounced for the high level workload.

Conclusions:
All three components of the transactional WAS workload suffered in a similar way when z/VM real memory and z/VM processing power was constrained.

An unexpected behavior was that for the medium level workload, and when z/VM was configured with 96 GiB real memory or less, the WAS CPU load increased as z/VM real memory was reduced.

Memory usage and paging workload analysis for CID 64/20

As with the database BI workload, here the detailed memory usage and workload analysis is only presented for the z/VM with a CID 64/20 configuration.

Figure 18 shows the average page distribution and the average page read rates for the medium level transactional WAS workload when running as part of the combined workload set in a z/VM CID 64/20 system configuration.

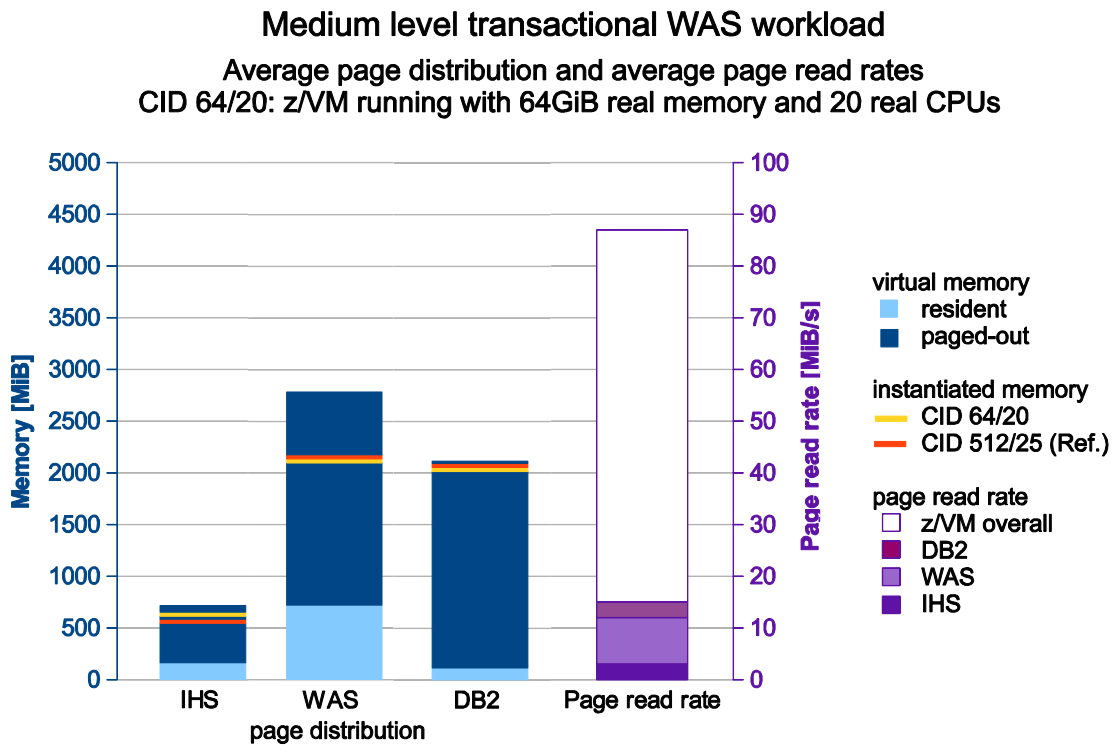

Figure 18: Medium level transactional WAS workload within CID 64/20: Average page distribution and average page read rates

Figure 19 shows the average page distribution and the average page read rates for the high level transactional WAS workload.

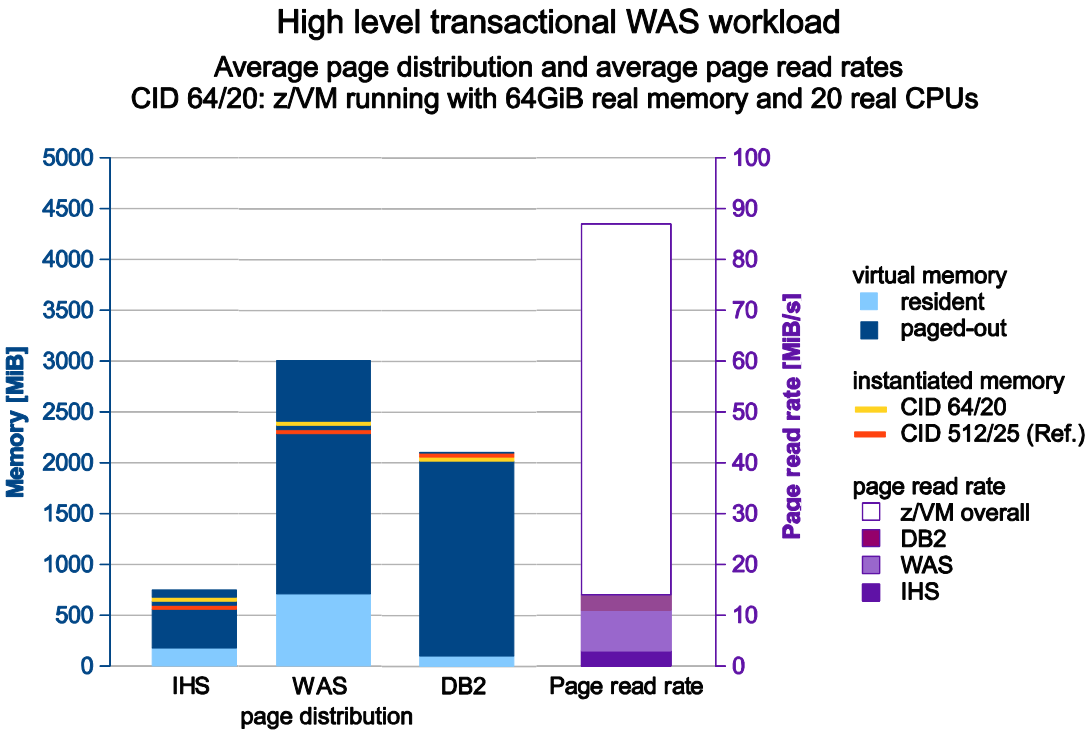

Figure 19: High level transactional WAS workload within CID 64/20: Average page distribution and average page read rates

A general description of the elements shown in Figure 18 and Figure 19 is given in *General description of page distribution and page read rate diagrams* (page 38). Note that the memory scale in Figure 18 and Figure 19 is different from that used in previous page distribution and page read diagrams.

Observations:
On average, for the IHS component, about one quarter of the instantiated memory was backed by resident memory, for the WAS component about one third was backed, and for the DB2 component only about 6 % were backed. Together, the three components of the transactional WAS workload occupied about 1.5 % of the z/VM DPA.

On average, the three components of the transactional WAS workload accounted for about one sixth of the z/VM page read operations. The averaged fractions of resident memory being paged in were about 0.4 % for the IHS component, 0.45 % for the medium level WAS component (0.34 % for the high level variant), and about 0.15 % for the DB2 component.

For the WAS component, the sum of the amounts of resident memory and stored memory was significantly higher than the instantiated memory, while for the other two components these numbers approximately match.

Within the transactional WAS workload, the WAS component exhibited the highest page read rate - about three times as high as each of the other workload components.

Conclusions:
The conclusions are presented separately for each workload component:

- **IHS**
  Previous experiences show that IHS usually exhibits moderate virtual memory requirements, using a few small application buffers accumulating to about 160 MiB, but no large buffers. In context of the DayTrader workload,

it mainly acts as a gateway server, more or less passing through most incoming requests to the WAS server. In contrast to a typical web server environment, in this context the IHS server does *not* have to provide access to large amounts of web pages stored in files. Consequently, the use of the page cache is limited, as only few file accesses result. As a result, the memory footprint of the IHS server is small - around 500 MiB. Note that some of this memory usage is imposed by the Linux system itself.

- **WAS**
  From the three components of the transactional WAS workload, the WAS component is the most active virtual memory consumer. The resident fraction of the instantiated memory for the WAS component is also the largest of all components, indicating intense use of memory. This is also indicated by the highest page-in rate of all components.

  The dominant part of the virtual memory allocated in the WAS server is the Java heap with a fixed size of 2 GiB. Particularly within the high level workload, the instantiated memory of the WAS component is larger than the allocated Java heap, indicating that additional virtual memory is needed for elements outside of the Java heap.

  The actual usage of the memory from the Java heap can be estimated by inspecting Linux anonymous memory (as reported in the Linux proc file system under /proc/meminfo). The amount of Linux anonymous memory was 1.2 GiB for the medium level workload, and 1.6 GiB for the high level workload. This indicates that in both cases the Java heap was probably not expanded to its configured size. Nevertheless, workload processing was performed flawlessly even when only about half of the Java heap was kept resident, albeit with performance degradations.

  Furthermore, the use of the gencon garbage collection policy might have been a factor for the transactional WAS workload being able to tolerate the impact of moderate z/VM real memory constraints. This policy tries to reduce the amount of pages modified during a garbage collection.

- ***DB2***
  It is assumed that DB2 allocates the dominant part of the virtual memory as Linux shared memory for database buffers. The amount of allocated shared memory is around 1.3 GiB for both workload levels (as reported by the Linux sar report).

  Within the DB2 virtual system, apparently all virtual memory is allocated - the amount of instantiated memory equals the virtual memory size. This might be attributed to DB2 using page-cached I/O that can cause large regions of virtual memory to be allocated. However, DB2 does not make the same intense use of memory as the other two components - the resident fraction of its instantiated memory is much smaller. At the same time, it causes a page-in rate similar to that of the IHS component. This might indicate that the memory usage of the DB2 component is less frequent, but similar to that of the IHS component.

  A factor influencing the DB2 virtual memory access behavior is that DB2 is the only component doing disk I/O, predominantly write operations. Page-cache based write operations imply that involved virtual memory pages are dirtied, which in turn invalidates corresponding previous copies of these pages that z/VM maintains in its paging store.

  Finally - as already observed with the database BI workload - a database is a highly sophisticated system that is able to attenuate the impacts of memory latencies by caching data in buffer pools, and by scheduling asynchronous operations.

File system I/O workload

This section describes the analysis of the test results from processing the file system I/O workload under various constrained z/VM configurations.

Workload analysis

Figure 20 shows the relative throughput of the medium level file system I/O workload for both the direct I/O and the page-cached variant, when these were executed as part of the combined workload set within a variety of differently constrained z/VM configurations.

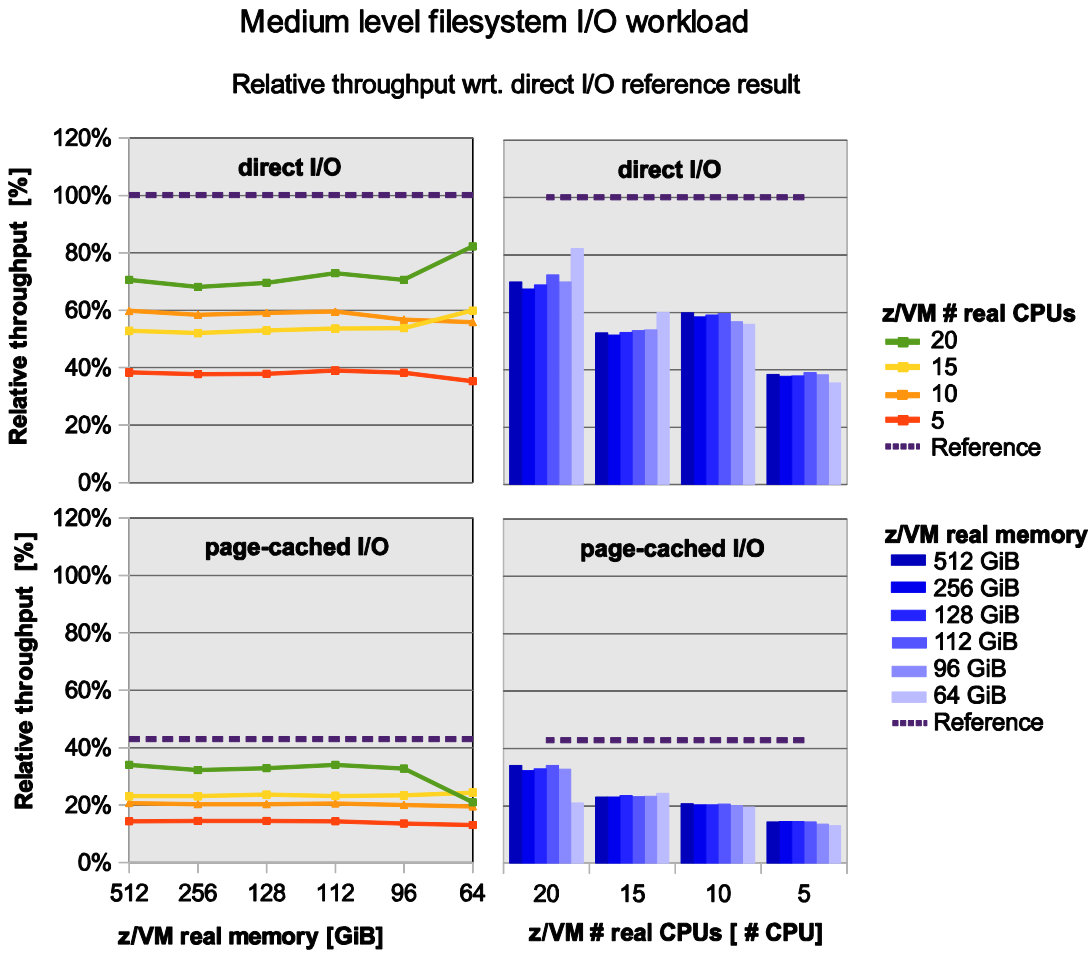

Figure 20: Medium level file system I/O workload: Relative throughput wrt. direct I/O reference result

Figure 21 shows the relative throughput of the high level file system I/O workload for both the direct I/O and the page-cached variant.

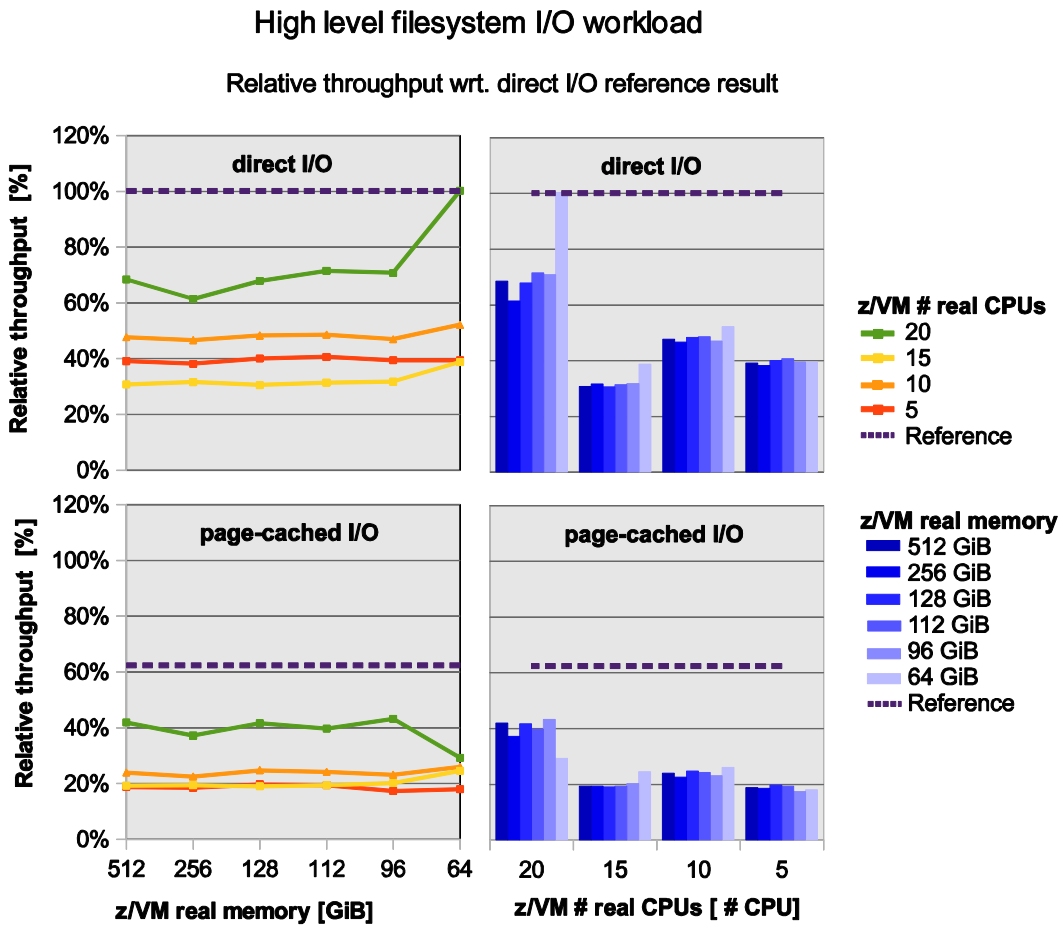

Figure 21: High level file system I/O workload: Relative throughput wrt. direct I/O reference result

A general description of the elements shown in Figure 20 and Figure 21 is given in *General description of relative throughput diagrams* (page 35).

Observations:
The dependency of the throughput on the amount of z/VM real memory remained in the area of a few percent, except when z/VM had 20 real CPUs configured. In this case, and when z/VM real memory was 64 GiB or less, the throughput of the direct I/O variant increased in spite of z/VM real memory being more constrained. At the same time, the throughput of the page-cached variant decreased respectively. In fact, the throughput achieved by the high level workload in this case even slightly exceeded that of the unconstrained case.

Conclusions:
The file system I/O workload did not benefit from using the page-cache. When z/VM real memory was significantly constrained, but processing power was not constrained, the direct I/O variant apparently even reclaimed throughput at the expense of the page-cached variant.

A possible explanation for this at first unexpected behavior might be higher virtual memory access latencies. As page-cache pages were dirtied by fio writes, but were not immediately written to the target file by Linux, the access rate on such pages might have appeared low to z/VM. When under real memory pressure, z/VM very soon might have decided to page-out such pages, in order to free corresponding page frames for use by other memory consumers. However, later, when Linux again accessed such pages in order to write them to the I/O file, z/VM first had to obtain (or reclaim) page frames, and fetch previously paged out pages.

A central point in favor of the direct I/O variant is that it uses only a few and comparatively small data transfer buffers. However, these buffers are used very intensly, preventing that respective virtual memory is paged-out by z/VM. Consequently, the workload is not hindered by memory access latencies and can continue to make use of CPU resources.

Another factor working in favor of the direct I/O variant is the fact that the target I/O resources for both file system I/O workload variants were located on the same storage server, and were connected by the same SAN resources (see Hardware). As the server that was processing the page-cached variant became more and more limited in accessing pages involved in page-cache I/O operations, at the same time I/O resources became relatively better available for the server that was processing the direct I/O variant.

On the other hand, if in addition the processing power was limited for z/VM and implicitly also for the virtual systems running under z/VM, then, as a result, the process of dirtying pages as well as any other virtual system activities was retarded. Insofar, the effects resulting from constrained z/VM real memory were moderate, the more the processing power became limited, or, in case of very constrained processing power, were not observed at all.

Memory usage and paging workload analysis for CID 64/20

As with the database BI workload, here the detailed memory usage and workload analysis is only presented for the z/VM configuration with 64 GiB real memory and 20 real CPUs (CID 64/20).

Figure 22 shows the average page distribution and the average page read rates for both levels of the page-cached variant of the file system I/O workload, when running as part of the combined workload set in a z/VM CID 64/20 system configuration.

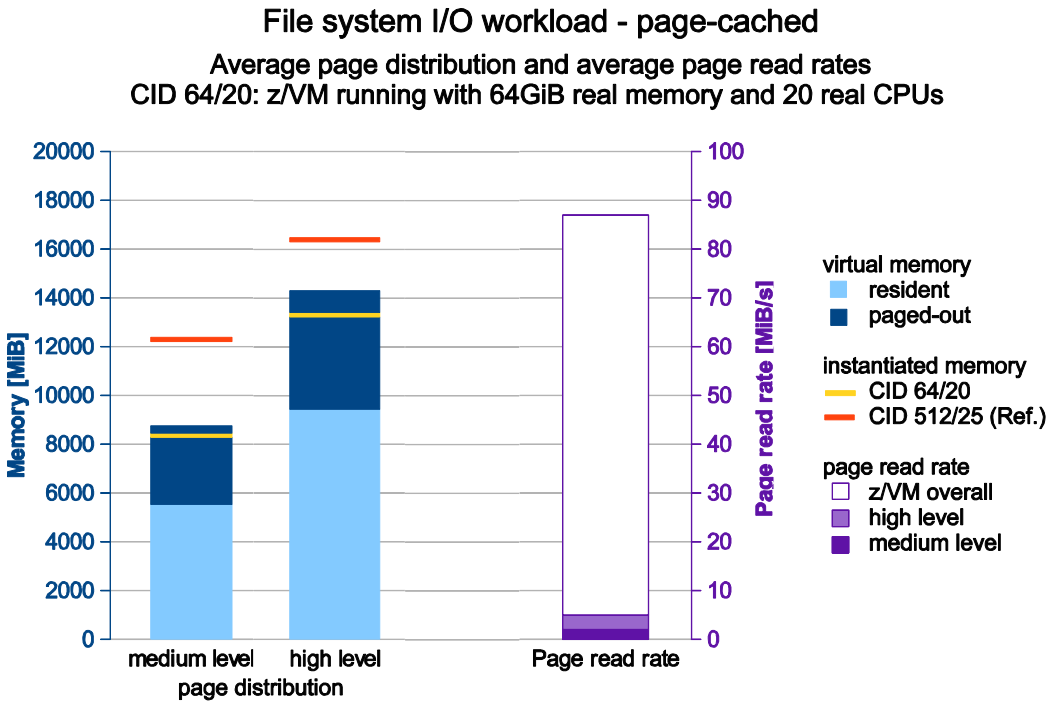

Figure 22: File system I/O workload - page-cached variant within CID 64/20: Average page distribution and average page read rates

A general description of the elements shown in Figure 22 is given in *General description of page distribution and page read rate diagrams* (page 38).

Observations for the page-cached variant of the file system I/O workload:
The page-cached variants of the file system I/O workload exhibited rather large amounts of instantiated memory. For the medium level workload, on average about 66 % of the instantiated memory was backed by resident memory, and for the high level workload, this percentage was even 71 %. This resident memory accounted for 8.5 % (medium level) and 14.5 % (high level) of the z/VM DPA. The average amounts of instantiated memory observed during the reference execution were significantly higher than those observed with the z/VM CID 64/20 configuration.

The page-cached variants of the file system I/O workload averaged at about 2.3 % (medium level) and 3.4 % (high level) of z/VM's page read operations. The averaged fractions of resident memory being paged in were about 0.4 % for the medium level, and about 0.3 % for the high level workload.

Figure 23 shows the average page distribution and the average page read rates for both levels of the direct I/O variant of the file system I/O workload. Note that the left scale for memory is different from that used in Figure 22.

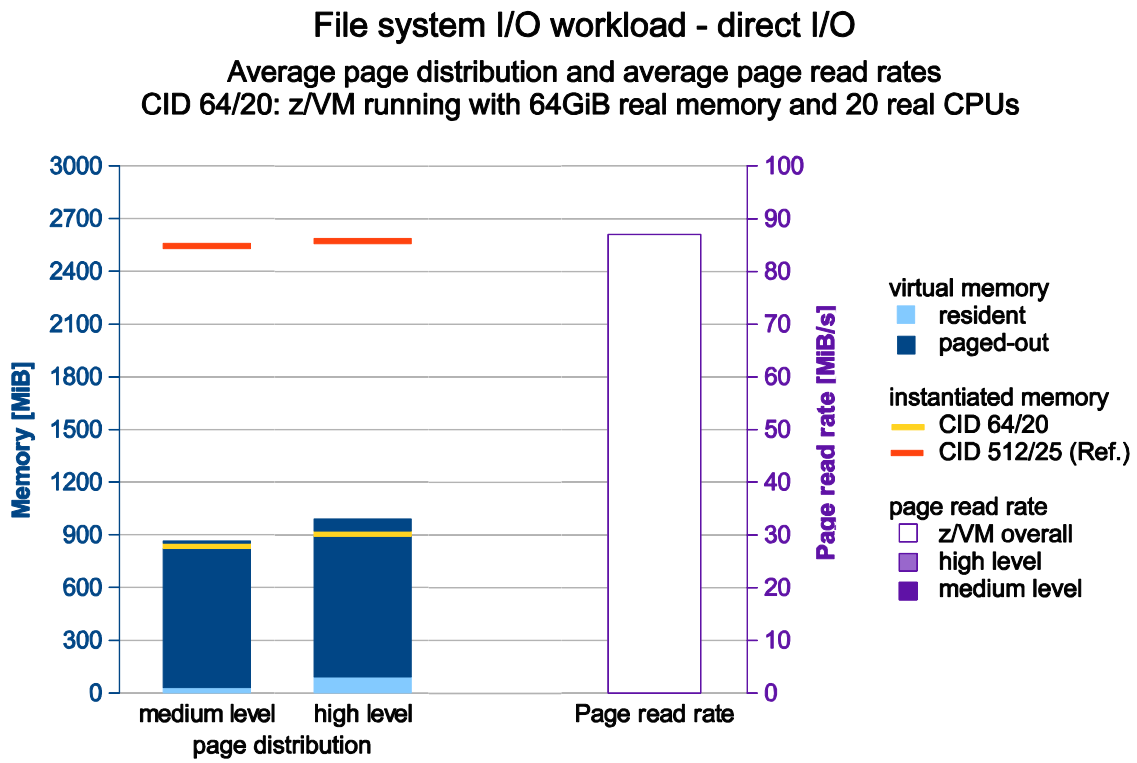

Figure 23: File system I/O workload - direct I/O variant within CID 64/20: Average page distribution and average page read rates

A general description of the elements shown in Figure 23 is given in *General description of page distribution and page read rate diagrams* (page 38). Note that the memory scale in Figure 22 and Figure 23 is different from that used in previous page distribution and page read diagrams.

Observations for the direct I/O variant of the file system I/O workload:
The direct I/O variants of the file system I/O workload exhibited rather small amounts of instantiated memory. For the medium level workload, on average, about 3.6 % of the instantiated memory was backed by resident memory, and for the high level workload this percentage was 9.8 %. These resident memory sizes accounted for 0.05 % (medium level) and 0.14 % (high level) of the z/VM DPA. Again, the average amounts of instantiated memory observed during the reference execution were significantly higher than those observed with the z/VM CID 64/20.configuration.

The direct I/O variants of the file system I/O workload did not cause any significant z/VM paging activities.

Conclusions for both variants of the file system I/O workload:
As expected, when using the page-cache for the file system I/O workload, much higher memory consumption results. In this case, when memory constraints at the z/VM level result in respective paging activities, memory access delays are imposed at the virtual system level.

On the other hand, the direct I/O variants of the file system I/O workload used only small amounts of memory. As memory constraints at the z/VM level became more distinct, the direct I/O variants might have profited from other workloads not reaching their full processing capacities because they were more affected by memory access latencies.

The fact that the average amounts of instantiated memory observed within CID 64/20 were much smaller than those observed during the reference execution could be explained as follows:

Any test case execution went through an initial phase, during which the software within each virtual system was initialized for the test. During this phase, typically memory intense tasks had to be performed. After this initial phase, Linux flagged unused parts of the virtual memory as unused (see also Interactions between virtual memory consumers and providers).

When executing within a configuration that is not memory constrained - such as during the reference execution - CP is not under pressure to reassign real memory backing unused virtual memory for other purposes. Thus, in this situation, the unused virtual memory could remain part of the instantiated memory.

However, when executing within a memory constrained environment such as within CID 64/20, CP removes these virtual memory parts from the virtual system's instantiated memory as soon as it detects that parts of virtual memory were declared unused by virtual systems. CP can then reassign the backing real memory for other purposes. Of course, this approach is preferable over stealing the backing real memory that is in use, because used virtual memory would have to be paged-out before the backing real memory could be reassigned.

With respect to the allocation and use of virtual memory there is a major difference between the page-cached variant and the direct I/O variant of the workload:

- **Page-cached I/O variant**
  With this variant the data relating to file I/O operations is cached in the page cache. The observed size of the page cache was about 6 GiB for the medium level variant, and about 10 GiB for the high level variant. Note that the page cache virtual memory requirements are in addition to other memory usage, such as that for application buffers or for device driver buffers.

  The lower amount of instantiated memory when compared to that in use during the reference execution is caused by the significantly reduced throughput that in turn caused a lower need for page cache space.

- **Direct I/O variant**
  With this variant the data relating to file I/O operations is directly transferred between application buffers and device driver buffers, without using the page cache. This results in a very small memory footprint. However, even here the observed size of the page cache was about 300 MiB. In this case the use of the page cache was imposed by other elements maintained by the Linux operating system, such as shared libraries or processes.

The virtual systems assigned for executing the file system I/O workload are examples of memory oversized virtual systems. Nevertheless, this does not cause a significant waste of real memory, because real memory backing unused virtual memory can be reassigned for other purposes when needed.

Java workload

This section describes the analysis of the test results from processing the Java workload under various constrained z/VM configurations.

Workload analysis

Figure 24 shows the relative throughput of the medium level Java workload when processed as part of the combined workload set within a variety of differently constrained z/VM configurations.

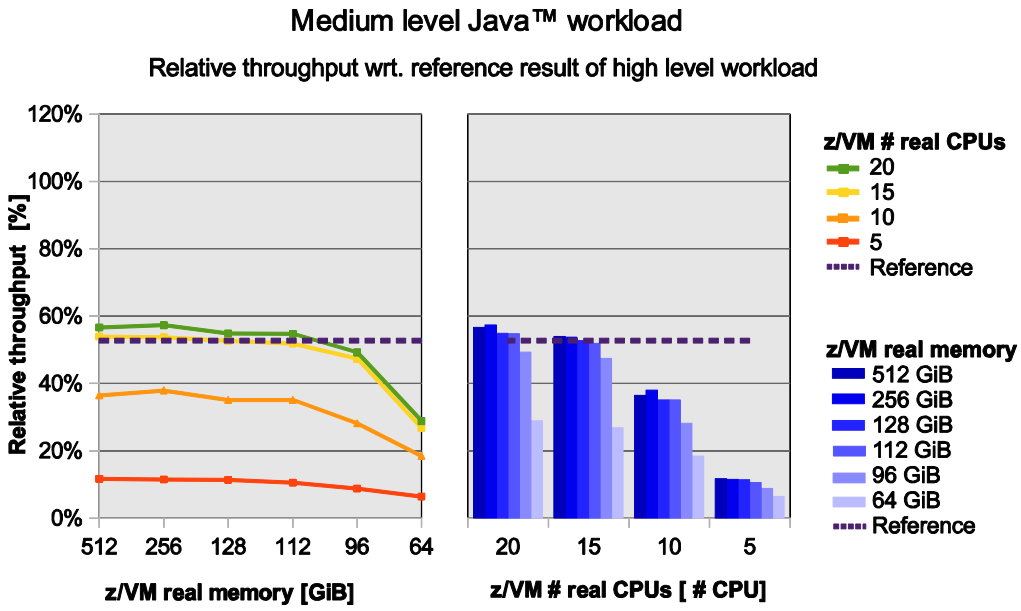

Figure 24: Medium level Java workload: Relative throughput wrt. reference result of the high level workload

Figure 25 shows the relative throughput of the high level Java workload.

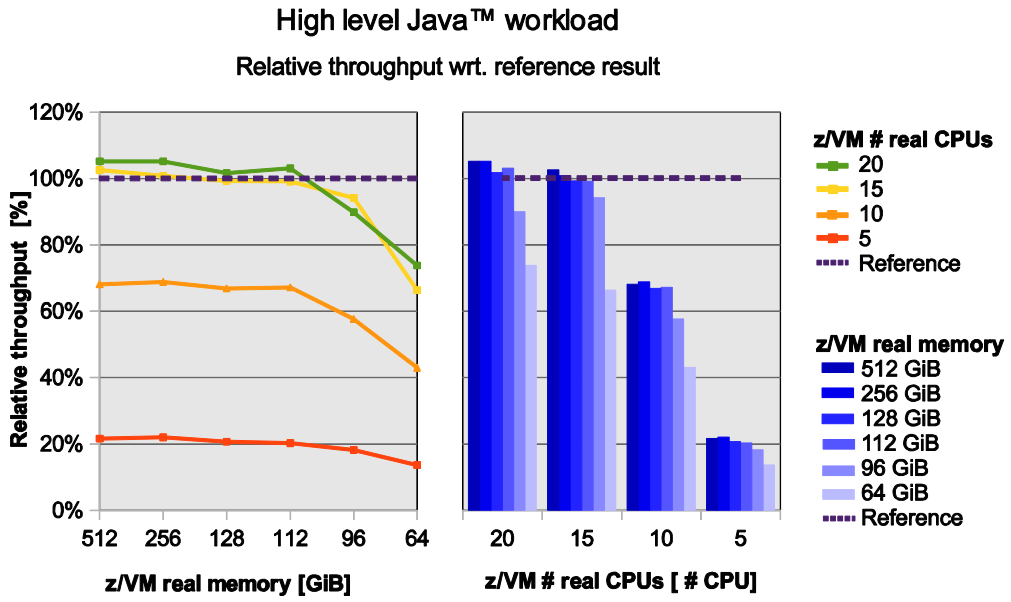

Figure 25: High level Java workload: Relative throughput wrt. reference result

A general description of the elements shown in Figure 24 and Figure 25 is given in *General description of relative throughput diagrams* (page 35).

Observations for both levels of the Java workload:
When running during the reference execution, the relative throughput of the medium level Java workload was about 53 % of that of the high level workload.

When less z/VM real memory was configured, the influence on the throughput of the Java workload was mostly insignificant as long as 112 GiB real memory or more were configured for z/VM. In these cases, when 15 or more real CPUs were configured for z/VM, the throughput was even slightly higher than that achieved during the reference execution. However, in any case, the throughput decreased considerably when 96 GiB real memory or less were configured for z/VM. The decrease was more pronounced for the medium level workload.

The throughput of the Java workload was only slightly affected by the number of real CPUs configured for z/VM, when 15 or more real CPUs were configured. When 10 or less real CPUs were configured for z/VM, the throughput declined sharply, with the decline being larger when the number of real CPUs was reduced from ten to five.

Conclusions for both levels of the Java workload:
The Java workload seems likewise to be impacted by z/VM real CPU and real memory constraints. While the impact stayed low, as long as 112 GiB real memory or more, and 15 real CPUs or more, were configured for z/VM, it became significant as the resources were configured below these limits.

On the positive side, when z/VM real CPU and real memory constraints were small, the Java workload could apparently profit, whereas other workloads were more seriously impacted by these small constraints.

Overall, the Java workload seems capable to tolerate moderate resource constraints at the z/VM level, but the throughput deteriorates when the resource constraints exceed a certain level. This limit needs to be detected by performance analysis. In our environment, we found that 112 GiB real memory or more, and 15 real CPUs or more need to be configured for z/VM in order to retain acceptable throughput of the Java workload.

CPU usage analysis
Figure 26 shows the average CPU load for the medium level Java workload when running as part of the combined workload set within a variety of differently constrained z/VM configurations.

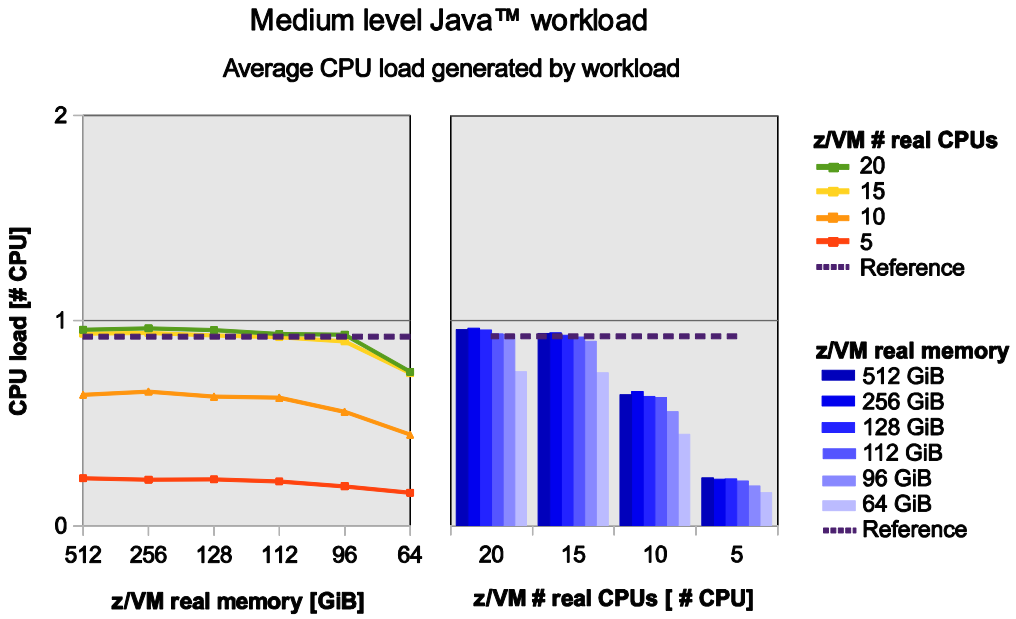

Figure 26: Medium level Java workload: Average CPU load generated by the workload

Figure 27 shows the average CPU load for the high level Java workload.

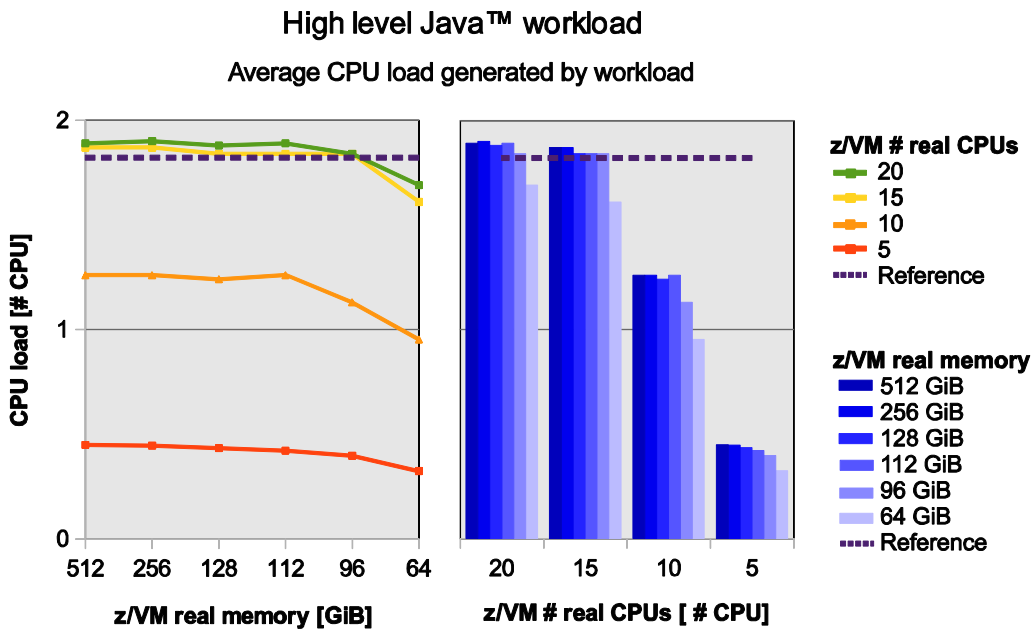

Figure 27: High level Java workload: Average CPU load generated by the workload

A general description of the elements shown in Figure 26 and Figure 27 is given in *General description of relative throughput diagrams* (page 35), except that in Figure 26 and Figure 27 CPU load is presented instead of throughput.

Observations for both levels of the Java workload:
The CPU load generated by the Java workload was almost independent from the amount of z/VM real memory, when 112 GiB real memory or more were configured for z/VM. When the amount of real memory configured for z/VM was below 112 GiB, a small decline of the CPU load resulted.

The amount of real CPUs configured for z/VM had almost no influence on the CPU load generated by the Java workload, when 15 or more real CPUs were configured for z/VM, but declined more than linearly as less than 15 real CPUs were configured for z/VM. This effect was even more significant for the medium level workload.

Conclusions for both levels of the Java workload:
As already diagnosed in the Workload analysis, when running as part of the combined workload set, the Java workload reacted lightly and in some cases even positively on small z/VM resource constraints, but reacted strongly, as the z/VM resource constraints became severe.

Consequently, when executing a Java workload within a z/VM environment where occasional real resource constraints are possible, the break point of the performance of the Java workload should be determined by an initial performance analysis, and during later processing should be monitored in order to not exceed that break point. If the break point is reached or exceeded, the temporary addition of real CPUs to the z/VM configuration might help to overcome a Java workload throughput degradation.

Memory usage and paging workload analysis for CID 64/20
As usual, the detailed memory usage and workload analysis is only presented for the z/VM configuration with 64 GiB real memory and 20 real CPUs (CID 64/20).

Figure 28 shows the average page distribution and the average page read rate for both levels of the Java workload when executed as part of the combined workload set, with z/VM running as a CID 64/20 configuration.

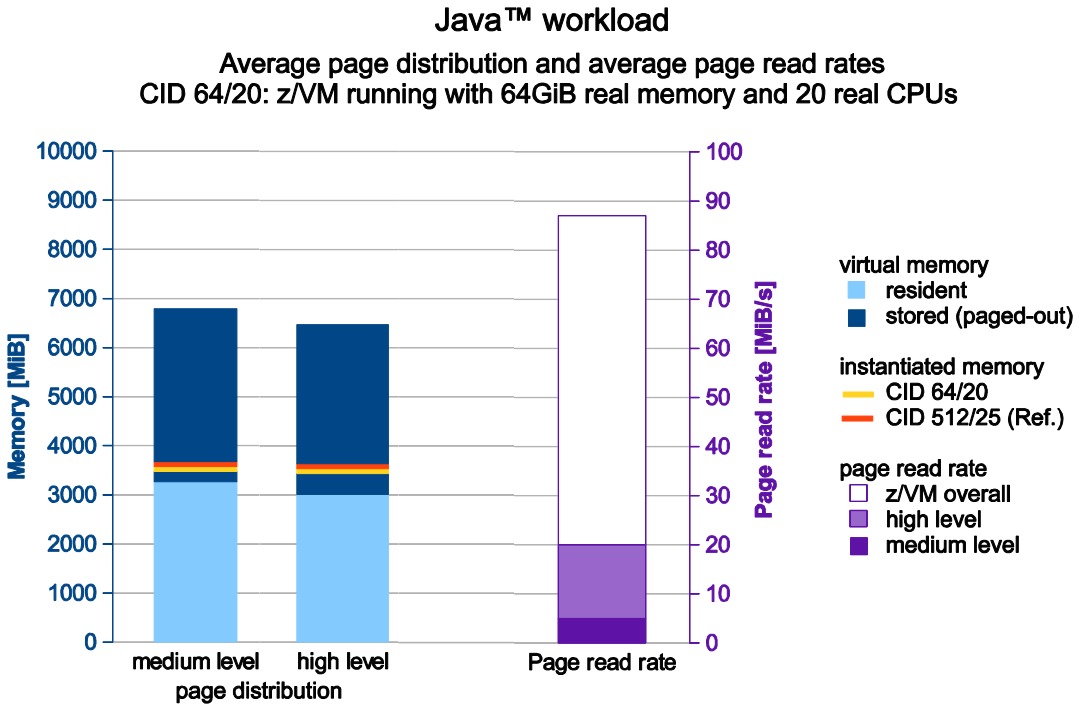

Figure 28: Java workload within CID 64/20: Average page distribution and average page read rates

A general description of the elements shown in Figure 28 is given in *General description of page distribution and page read rate diagrams* (page 38). Note that the memory scale in Figure 28 is different from that used in previous page distribution and page read diagrams.

Observations:
Both levels of the Java workload exhibited rather large amounts of instantiated memory that were mostly backed by resident memory. For the medium level workload, on average about 91 % of the instantiated memory was backed by resident memory, and for the high level workload this percentage was 85 %. These resident memory sizes accounted for 5 % (medium level) and 4.6 % (high level) of the z/VM DPA. In both cases, the sum of resident memory and stored memory is much larger than instantiated memory, indicating that a large number of pages is kept in both places, real memory and on paging storage.

The Java workload averaged at about 5.7 % (medium level) and 17.2 % (high level) of the z/VM page read operations. The averaged fractions of resident memory being paged in were about 0.15 % for the medium level, and about 0.5 % for the high level workload.

Conclusions:
The high fractions of resident memory indicate that the Java workload makes intense and widespread use of virtual memory. This observation is confirmed by the comparatively high page-in rates that - combined for both workload levels - make up for about a quarter of the total z/VM page read rate.

As detailed in the workload analysis, workload processing of the Java workload within CID 64/20 is degraded by about 45 % (medium level) and 26 % (high level). Apparently, the virtual system that is processing the high level Java workload configured with two virtual CPUs, is less affected by z/VM memory resource constraints. This could indicate that the availability of the second virtual processor is not only beneficial for achieving a higher throughput (about twice as high as that of the medium level workload), but also does help to attenuate z/VM memory resource constraints.

With respect to the allocation and use of virtual memory the dominant part is the Java heap size of 3.2 GiB. The `optthruput` garbage collection policy applied for this workload is known to cause much higher and more widespread memory accesses during garbage collection than the `gencon` garbage collection policy applied for the transactional WAS workload.

Network workload

This section describes the analysis of the test results from processing the network workload under various constrained z/VM configurations.

Workload analysis
Figure 29 shows the relative throughput of the medium level network workload when executed as part of the combined workload set within a variety of differently constrained z/VM configurations.

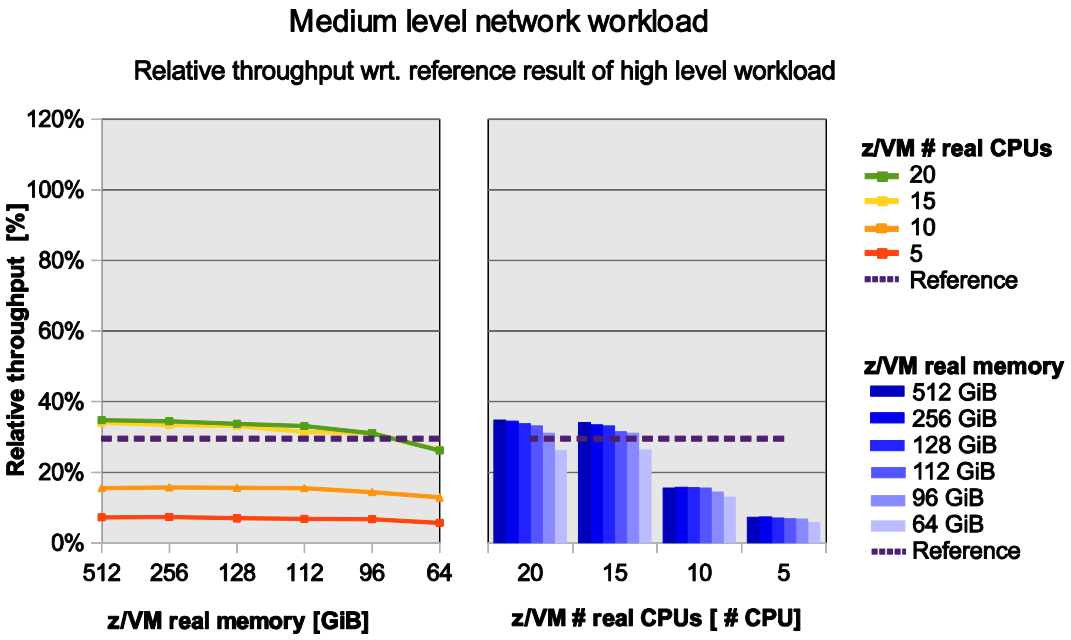

Figure 29: Medium level network workload: Relative throughput compared to the reference result of the high level workload

Figure 30 shows the relative throughput of the high level network workload.

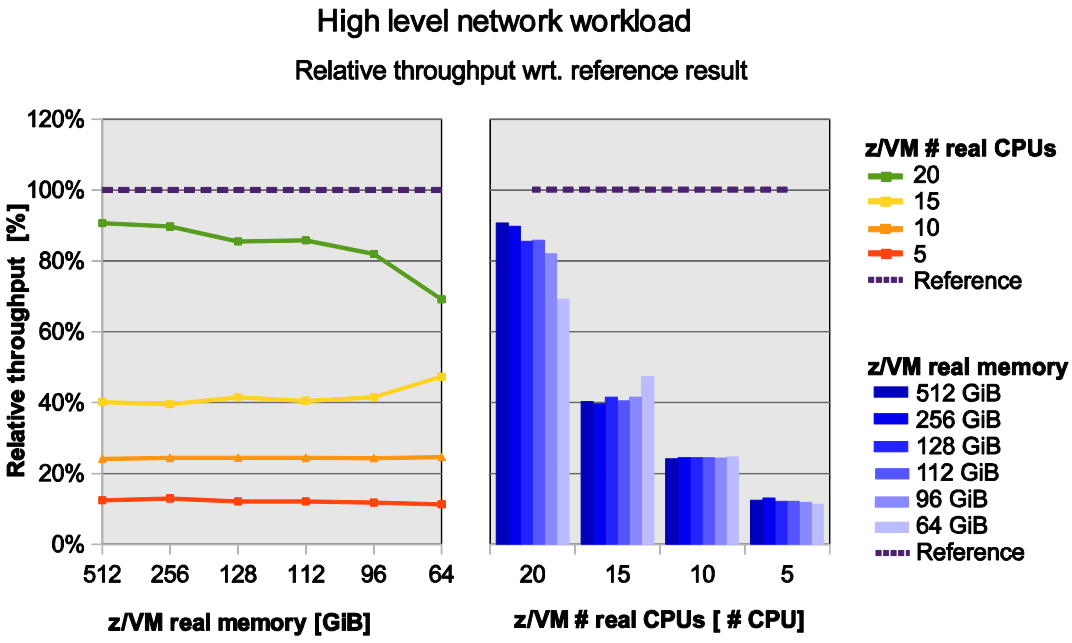

Figure 30: High level network workload: Relative throughput compared to the reference result

A general description of the elements shown in Figure 29 and Figure 30 is given in *General description of relative throughput diagrams* (page 35).

Observations for both levels of the network workload:
When processed during the reference execution, the relative throughput of the medium level network workload was about 30 % of that of the high level workload.

When less z/VM real memory was configured, the influence on the throughput of the network workload was mostly insignificant, except when 20 real CPUs were configured for z/VM, in which case the throughput slightly degraded as less real memory was configured for z/VM.

With 15 or more real CPUs, the throughput of the medium level network workload was consistently higher than that achieved during the reference execution, except when 64 GiB of real memory was configured for z/VM.

When 15 real CPUs were configured for z/VM, the high level network workload regained some throughput with 96 GiB real memory or less.

The throughput of the network workload was differently affected by the number of real CPUs configured for z/VM, depending on the workload level:

- The medium level network workload achieved better than reference throughput with 15 or more z/VM real CPUs configured. The throughput significantly declined as 10 or less z/VM real CPUs were configured.
- Within any configuration, the high level network workload did not reach the throughput value achieved during the reference execution. The throughput break already occurred with 15 z/VM real CPUs configured, down to about 40 % of the reference throughput, and then about declining linearly with decreasing numbers of configured real CPUs.

Conclusions for both levels of the network workload:
The network workload is primarily impacted by z/VM real CPU constraints.

The virtual systems (client and server) configured with two virtual CPUs processing the high level network workload is capable to achieve three times the throughput of the virtual systems configured with one virtual CPU processing the medium level network workload. However, as soon as noticeable constraints are introduced at the z/VM level, throughput of the high level network workload degrades significantly. Opposed to that, as long as z/VM resource constraints are not too excessive, the virtual system that is processing the medium level network workload is able to retain and partially even exceed its reference throughput.

CPU usage analysis
Figure 31 shows the average CPU load for the medium level network workload when processed as part of the combined workload set within a variety of differently constrained z/VM configurations.

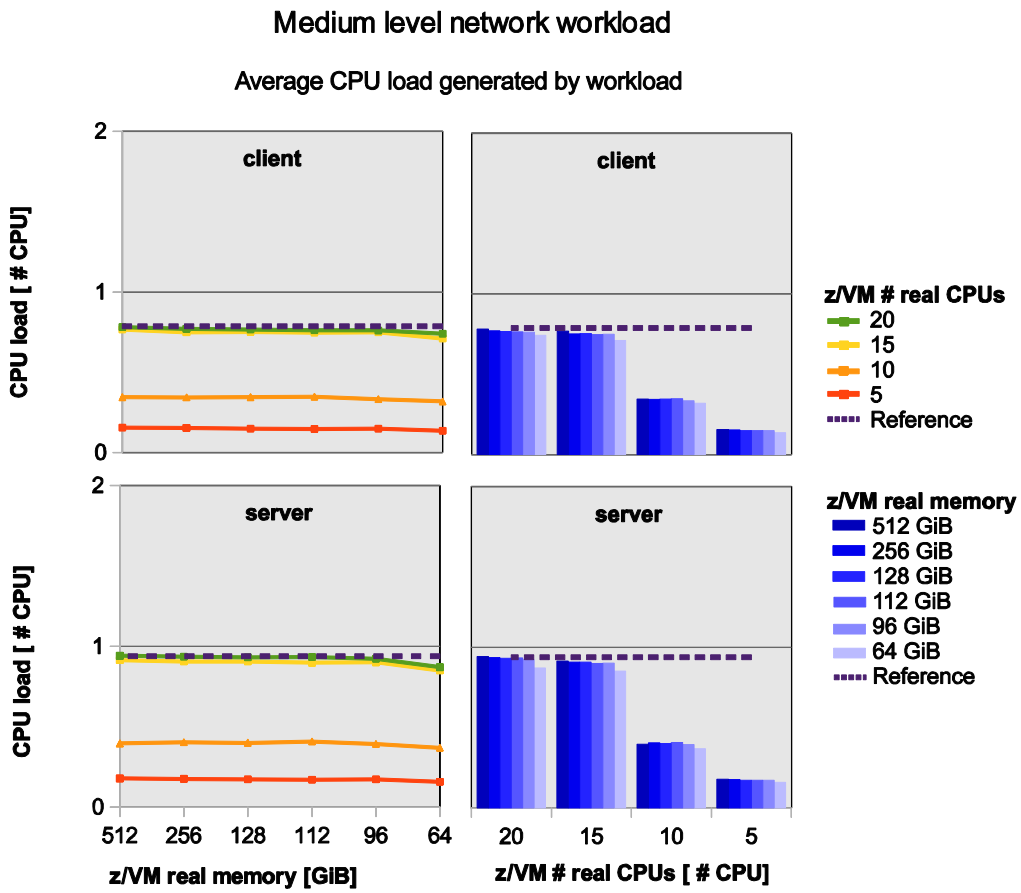

Figure 31: Medium level network workload: Average CPU load generated by the workload

Figure 32 shows the average CPU load generated by the high level network workload.

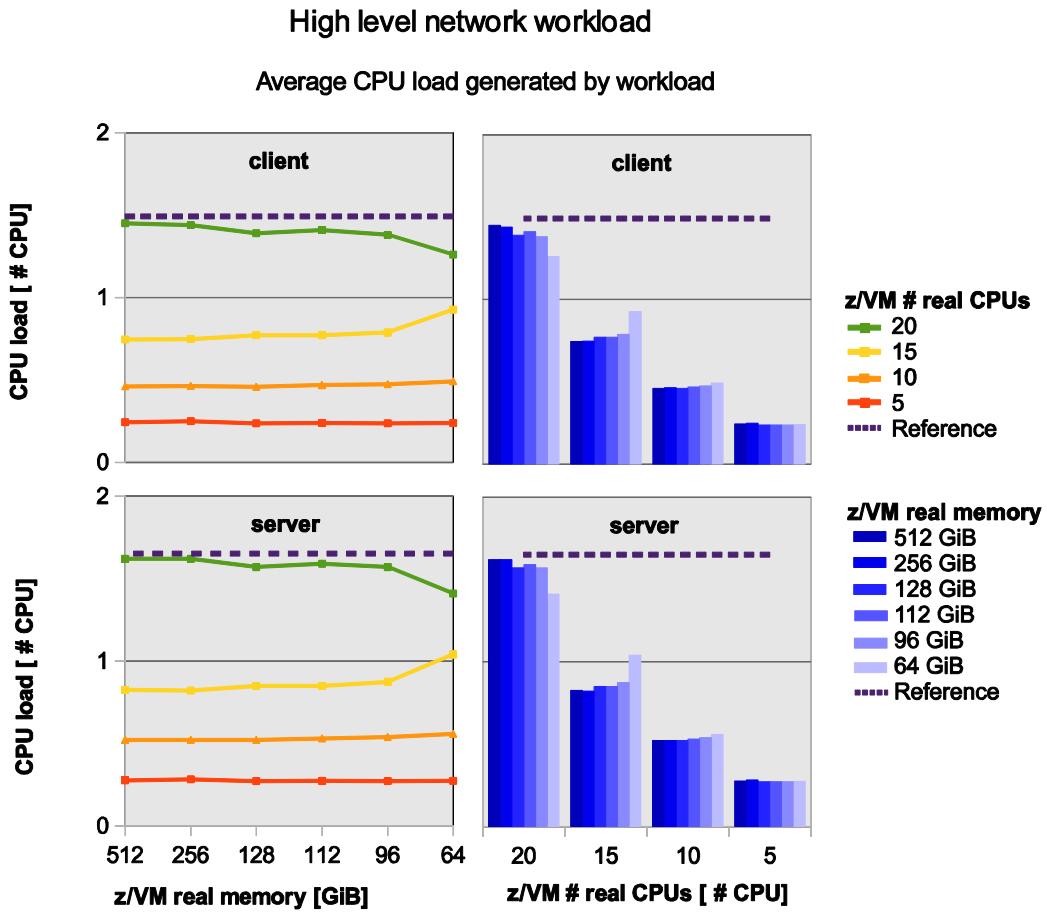

Figure 32: High level network workload: Average CPU load generated by the workload

A general description of the elements shown in Figure 31 and Figure 32 is given in *General description of relative throughput diagrams* (page 35), except that in Figure 31 and Figure 32 CPU load is presented instead of throughput.

Observations for both levels of the network workload:
The CPU load generated by the medium level network workload was mostly independent from the amount of z/VM real memory.

The CPU load generated by the high level network workload was mostly independent from the amount of z/VM real memory when 10 or less real CPUs were configured for z/VM. When 20 z/VM real CPUs were configured, the CPU load gradually declined with the amount of z/VM real memory, and when 15 real CPUs were configured, the CPU load gradually increased with the amount of z/VM real memory.

The CPU load generated by the server systems generally was a little bit higher than that generated by client systems, regardless of the workload level.

Conclusions for both levels of the network workload:
The CPU load analysis confirms the diagnosis of the workload analysis, that the throughput of medium level network workload is affected later than that of the high level network workload when real CPU constraints are introduced at the z/VM level. It seems that in z/VM real CPU constrained situations the high level network workload gained advantage with respect to CPU costs by being able to use a second virtual CPU.

Memory usage and paging workload analysis for CID 64/20
As usual, the detailed memory usage and workload analysis is only presented for the z/VM configuration with 64 GiB real memory and 20 real CPUs (CID 64/20).

Figure 33 shows the average page distribution and the average page read rate for the medium level network workload when processed as part of the combined workload set when z/VM is running in a CID 64/20 configuration.

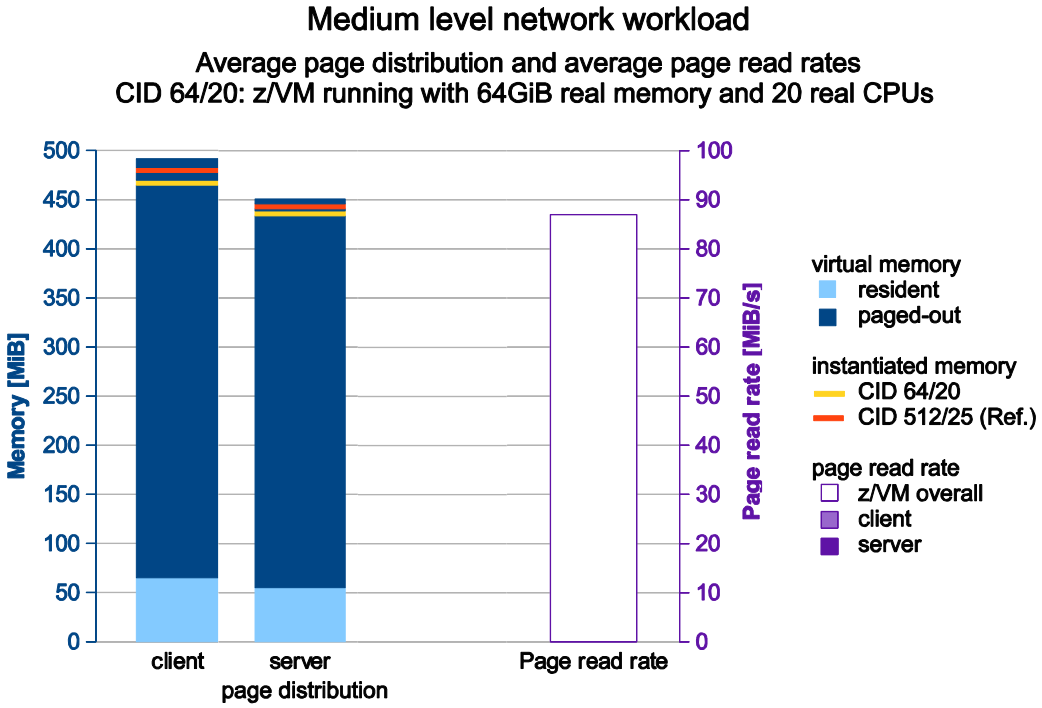

Figure 33: High level network workload within CID 64/20: Average page distribution and average page read rates

Figure 34 shows the average page distribution and the average page read rate for the high level network workload.

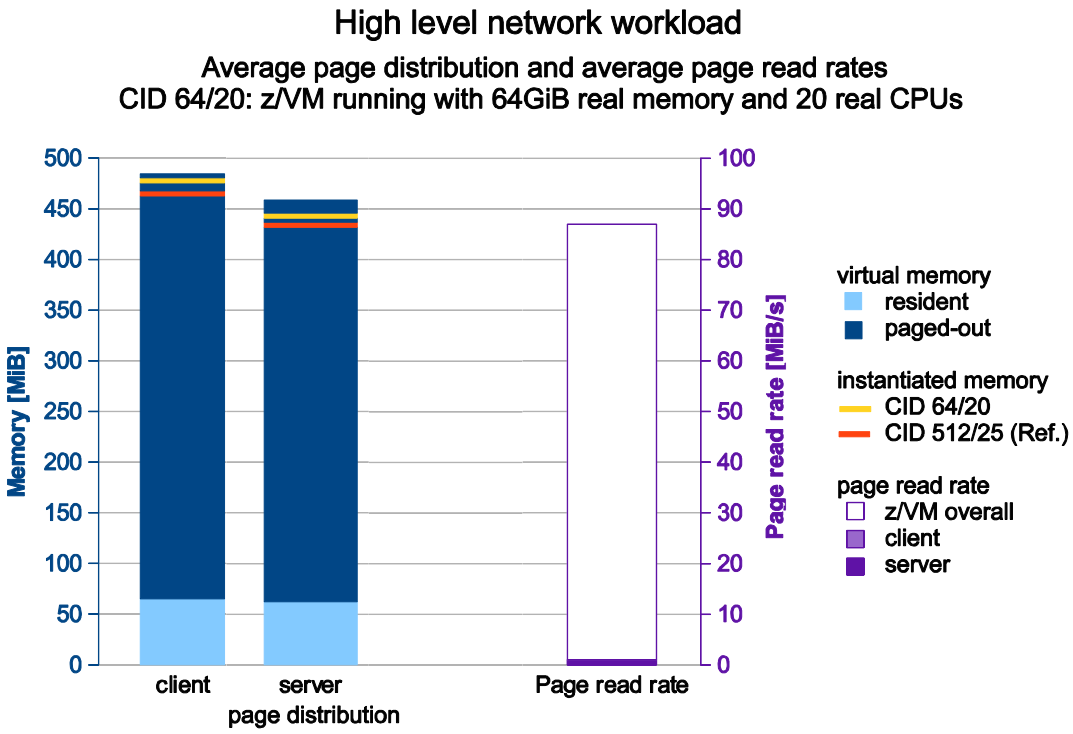

Figure 34: High level network workload within CID 64/20: Average page distribution and average page read rates

A general description of the elements shown in Figure 33 and Figure 34 is given in *General description of page distribution and page read rate diagrams* (page 38), except that in Figure 33 and Figure 34 CPU load is presented instead of throughput.

Observations for both levels of the network workload:
Both levels of the network workload exhibited a rather small amount of instantiated memory that was backed by resident memory only to a small fraction (about 13-14 %).

The z/VM paging activity caused by the network workload was negligible.

Conclusions for both levels of the network workload:
The network workload is not a big memory consumer. Nevertheless, the high level network workload was affected by z/VM memory constraints when z/VM was configured with 15 or 20 real CPUs.

A possible explanation for the low memory consumption of the network workload might be that the workload only makes use of small data buffers. These buffers are initially allocated once, and then repeatedly used for sending and receiving data, that is, for transferring data to/from the Linux device drivers. In that respect the network workload behaved similar to the direct I/O variant of the file system I/O workload.

Once received, data is discarded and not copied for further use, as it would be the case in a real application. Insofar, the results obtained for the network workload should be considered as a metric for the overhead that results from transferring data between virtual systems, but not as a metric that also covers the further processing of the transferred data.

With that in mind, the high level network workload might be a good candidate for reserving memory by means of the z/VM `set reserved` command, in order to support z/VM in keeping most of the small but highly used data transfer buffers in z/VM real memory.

References

View a list of documents referenced in this white paper.

- z/VM 6.3 Performance Considerations, IBM Corporation accessible from the z/VM Performance Website: http://www.vm.ibm.com/perf/reports/zvm/html/630con.html
- z/VM 6.3 CP Planning and Administration, SC24-6178-05, IBM Corporation accessible from the IBM Library Server http://publib.boulder.ibm.com/cgi-bin/bookmgr/download/HCSG0C20.pdf
- z/VM 6.3 Performance, SC24-6208-04, IBM Corporation accessible from the IBM Library Server http://publib.boulder.ibm.com/cgi-bin/bookmgr/download/HCSI1C20.pdf
- z/VM 6.3 Performance Toolkit Reference, SC24-6210-03, IBM Corporation accessible from the IBM Library Server http://publib.boulder.ibm.com/cgi-bin/bookmgr/download/HCSL7C20.pdf
- Linux on System z: Device Drivers, Features, and Commands (Kernel 3.14), SC33-8411-24, IBM Corporation accessible from the IBM developerWorks® Website: http://public.dhe.ibm.com/software/dw/linux390/docu/l314dd24.pdf
- DayTrader - a more complex application, Apache Foundation accessible from Apache Geronimo Website: http://geronimo.apache.org/GMOxDOC22/daytrader-a-more-complex-application.html
- fio readme, Jens Axboe accessible from the Git repository: https://github.com/axboe/fio
- SwingBench 2.2 Reference and User Guide (stable version 2.4.0.845 - Updated: 2011-12-08), Dominic Giles accessible from: http://www.dominicgiles.com/swingbench.html
- uperf - A network performance tool, Performance Applications Engineering group at Sun Microsystems Documentation and download: http://www.uperf.org/

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

COPYRIGHT LICENSE:

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

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information in softcopy, the photographs and color illustrations may not appear.

IBM Systems and Technology Group
Route 100
Somers, New York 10589
U.S.A.
Produced in the United States of America,
00/2014

ZSW03269-USEN-00