January 2012
Linux on IBM System z

WebSphere Application Server Horizontal Versus Vertical JVM Stacking Report

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

Table of Contents

Introduction

This report presents performance data for a test that compares the effects of WebSphere Application Server vertical JVM stacking to horizontal JVM stacking.

Customers have often asked for guidance on how best to implement a WebSphere Application Server multiple server environment. The question is whether it is better to host all WebSphere Application Servers on a single (vertical JVM[1] stacking) z/VM® guest, or distribute the servers across multiple guests (horizontal JVM stacking). In the past, it has been found that two portal servers on the same z/VM guest provided better performance than having them distributed across two z/VM guests.

There are two mechanisms that work against each other:

- The management overhead in z/VM scales with the number of guests.
- The management overhead and resource competition in Linux® scales with the number of JVMs per Linux instance.

This means, for example, that having all JVMs in one Linux guest provides the lowest z/VM overhead, but the highest overhead in Linux. But one JVM per guest and many guests result in the opposite behavior. The question is which of these mechanisms has the higher impact, and are there are other, currently unknown, factors that also have an impact on the system. The expectation is that the optimal scenario is somewhere in between the extreme ends.

This report gives the results for the following environments:

- All JVMs in one guest (full vertical stacking)
- One JVM in each guest, and many guests (full horizontal stacking)
- Configurations in between

The workload used in driving the various WebSphere Application Server environments was a simple web services application that did not require any database.

The number of JVMs (200) used in the test was held constant. With only one guest in the configuration, the 200 WebSphere Application Servers (JVMs) are defined on one guest. With two guests, each guest runs 100 WebSphere Application Servers. As the number of guests are scaled, the number of WebSphere Application Servers (JVMs) defined per guest decreases. However, the total number of WebSphere Application Servers (JVMs) under test is always held at 200.

As an additional test, the use of shared mini-disks or a DCSS for the WebSphere installation was analyzed, to determine if there are criteria that favor one or the other setup.

[1] JVM - Java Virtual Machine, which is the core of one WebSphere Application Server instance.

Summary

In the workload used in these tests, there appears to be little difference in throughput between Vertical WebSphere® JVM stacking or Horizontal WebSphere JVM stacking.

However, LPAR CPU utilization is a different story. Some configurations seem to consume less CPU resources than others, where both ends of the scale are high in CPU load.

- The lowest load was observed at 10 JVMs per guest (20 guests)
- The highest load was observed at 200 JVMs per guest (one guest)

z/VM overhead is not a contributor to the observed higher LPAR CPU load. In spite of this, it was very impressive to see how easily z/VM was able to handle 200 guests under load. The increased load comes from Linux, WebSphere, and JAVA™. The number of virtual CPUs seems to determine how many JVM threads are actively consuming CPU resources.

From the test results, the following recommendations could be given:

1. In the case where the CPU load per guest is less than one CPU, it seems that the WebSphere Application Server runs more efficiently with two virtual CPUs than with one virtual CPU.
2. Do not size the guest with more virtual CPUs than required for one WebSphere instance. This might limit the number of JVMs stacked in one guest, and therefore avoid a shortage in guest CPU resources.
3. Running with CPU overcommitment had a lower impact. But when recommendation 1 would cause very high levels of CPU overcommitment (such as 16:1), it was found that it is better to use one virtual CPU for the WebSphere guests, instead of two.

The use of a DCSS for the WebSphere installation tree has a very low impact on throughput. But the use of a DCSS has a significant impact on CPU utilization. The additional effort related to using a DCSS makes it attractive only when the number of guests using the DCSS exceeds a certain number. This number was found to be 10 guests in the test environment. With fewer guests, the use of a shared minidisk instead of a DCSS consumes fewer CPU resources. In the case where the system runs with memory constraints, the DCSS might provide additional advantages in saving memory.

If you have large z/VM guests and questions about the use of SET REORDER for your system, contact IBM z/VM Level 2 for further analysis and help.

Test environment

The test environment used for the WebSphere Application Server JVM stacking tests consisted of IBM zEnterprise™ 196 hardware and various software.

Hardware

The tests were done on an IBM zEnterprise 196 (z196) processor. Table 1 lists the hardware used.

Table 1. Hardware used for WebSphere Application Server horizontal versus vertical stacking test

System	Operating System	Number of CPUs	Memory size	Network	Architecture
One LPAR	z/VM version 6.1	24	200 GB	▪ Two 1 Gb Ethernet ▪ 2 VSWITCH	IBM z196
1 - 200 Linux guests	SUSE Linux SLES 11 SP1	24 - 1 Virtual CPUs	200 - 1 GB	2 VSWITCH	z/VM guest
-			2 GB		DCSS
Total					
IBM z196	physical:	24	200 GB	Two 1 Gb Ethernet	IBM System z® LPAR
4 workload generators	Red Hat Enterprise Linux AS Release 4 U3	4	4 GB	One 1 Gb Ethernet	IBM System x®

The test environment was one LPAR on an IBM zEnterprise 196 with:

- 24 CPUs
- 200 GB memory
- 8 FICON® channels
- Two 1 Gb OSA Express 3 connections

Network setup

All guests and clients are connected to two VSWITCHs. Each VSWITCH is assigned one OSA port.

The workload drivers were evenly divided between VSWITCHs. Half the drivers were attached to half the WebSphere Servers through VSWITCH1, and the other half through VSWITCH2.

Storage server

The z/VM and Linux guest systems used in this study were allocated in an IBM System Storage® DS8000®, which used the IBM ECKD™ format and FICON channels.

Software

Table 2 lists the software used to test WebSphere Application Server horizontal versus vertical stacking.

Table 2. Software used for WebSphere Application Server horizontal versus vertical stacking test

Product	Version and release	Comments
SOA based workload generator	IBM internal	
WebSphere Application Server	7.1	Latest patch level
SUSE Linux	SLES 11 SP1	Latest patch level, (2.6.32.27-0.2.2.s390x.rpm)
z/VM	5.4 RSU 0902	Latest RSU

System setup

This section describes the steps necessary to set up the systems for the WebSphere Application Server JVM stacking tests.

Basic setup

- The LPAR size is the same in all test cases.
- The number of JVMs are the same in all test cases.
- The z/VM and Linux guests do not have any memory constraints. There is no memory overcommitment.
- Focus is on transactional throughput, as reported by the workload driver.
- The final results are expressed as throughput and total CPU utilization versus the number of JVMs per guest.

z/VM guest setup

Each z/VM Linux guest was defined on minidisks. The guest IDs used were LNX00001 through LNX00200. Table 3 shows the minidisk sizes used.

Table 3. z/VM guest minidisk sizes

Guest ID	Minidisk address	Minidisk size	Function
LNX00001 - LNX00002	▪ 100 ▪ 101 ▪ 102 ▪ 103 ▪ 107	▪ 60 ▪ 1609 ▪ 1669 ▪ 3338 ▪ 5008	▪ /boot ▪ /usr/share ▪ /usr/local ▪ / ▪ /opt/wasprofiles
LNX00003 - LNX00004	▪ 100 ▪ 101 ▪ 102 ▪ 103 ▪ 107	▪ 60 ▪ 1609 ▪ 1669 ▪ 3338 ▪ 1500	▪ /boot ▪ /usr/share ▪ /usr/local ▪ / ▪ /opt/wasprofiles
LNX00005 - LNX00200	▪ 100 ▪ 101 ▪ 102 ▪ 103 ▪ 107	▪ 60 ▪ 1609 ▪ 200 ▪ 3338 ▪ 1000	▪ /boot ▪ /usr/share ▪ /usr/local ▪ / ▪ /opt/wasprofiles
Shared minidisks	▪ 104 ▪ 105	▪ 5800 ▪ 200	▪ WebSphere binaries ▪ WebSphere .nif

The Linux guests were cloned from a master Linux image with the process outline in the following documents:

- Sharing and Maintaining SLES 11 Linux under z/VM using DCSSs and an NSS
  www.vm.**ibm.com**/linux/dcss/ror-s11.pdf
- z/VM and Linux on IBM System z, The Virtualization Cookbook for SLES 10 SP2
  www.redbooks.**ibm.com**/redbooks/pdfs/sg247493.pdf

WebSphere was installed and set up following the process outlined in:

- How to - Share a WebSphere Application Server V7 installation among many Linux for IBM System z
  www.**ibm.com**/support/techdocs/atsmastr.nsf/WebIndex/WP101817

Then the shared WebSphere installation was copied from the two mini disks to two DCSSs. The DCSSs were defined as follows:

```
cp defseg S11WAS70 3331f00-33a1eff sr loadnshr
cp defseg S11WASNF 33a1f00-33d1eff sr loadnshr
```

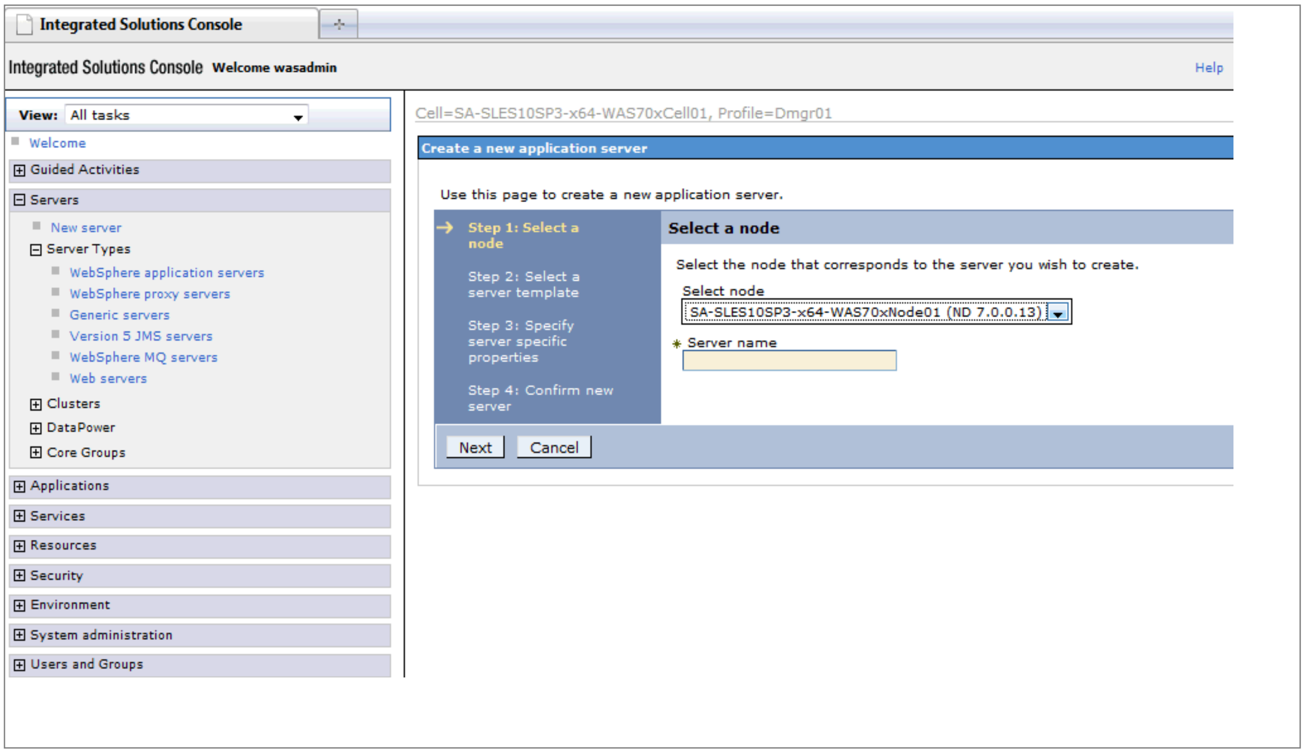
On the Linux guests, the DCSSs are mounted as read only files with the **-xip** (execute in place) option. This output is from the **mount** command:

```
/dev/dcssblk0 on /opt/IBM/WebSphere type ext2
(ro,noatime,nodiratime,xip,acl,user_xattr)
/dev/dcssblk1 on /opt/.ibm/.nif type ext2
(ro,noatime,nodiratime,xip,acl,user_xattr)
```

WebSphere Application Server setup

WebSphere Application Server, Network Deployment, Version 7.0 was used to create the test environment. Figure 1 illustrates the new server creation.

There is a one-to-one relationship between the node and application server profiles. When a profile is federated into a deployment manager cell, that profile becomes a node in the cell. Then, when another profile is federated into that same cell, that profile becomes a second and unique node in the cell, and so on. The deployment manager administrator console can be used to create new application servers, which become new nodes in the cell.



Cell=SA-SLES10SP3-x64-WAS70xCell01, Profile=Dmgr01

Create a new application server

Use this page to create a new application server.

Select a node

Select the node that corresponds to the server you wish to create.

Select node: SA-SLES10SP3-x64-WAS70xNode01 (ND 7.0.0.13)

* Server name

Figure 1. WebSphere Administrator console: New server creation

WebSphere administration scripts were used to create the required 200 nodes within a single deployment manager cell for a single z/VM guest. For multiple guest tests, the number of nodes required on each guest ranged form: 100, 50, 25, 20, 10, 4, 2, to 1. Each guest was set up as a unique deployment manager cell. WebSphere administration scripts were used to create the appropriate number of nodes within the cell.

Workload description

A SOA (Service Oriented Architecture) based workload was used in a stand-alone mode (no database required).

It models a motor insurance company. There are three major facets to the company infrastructure:

- Claims services (designed as Web Services)
- Call Center Claims application
- Third party gateway

For this measurement, the Claims services facet was used. This facet uses only the Web Services feature of WebSphere. The Claims services facet can use multiple payload sizes for both the request and response. This test used a payload size of 10 KB for both request and response.

Test methodology and scenarios

This section is an overview of how the WebSphere Application Servers are configured to compare vertical JVM stacking to horizontal JVM stacking. The test scenarios are also described.

These specific attributes and parameters were used in the test methodology:

- The WebSphere binaries are loaded from a shared DCSS mounted with the ***-xip*** option.
- Alternatively, WebSphere binaries are loaded from a shared minidisk.
- The physical resources in terms of processors, memory, network cards, and so on are kept constant throughout the test.
- The number of JVMs in the LPAR are kept constant throughout the test. The number of JVMs represents the total workload that the customer needs to run.
- The workload configuration per JVM is kept constant, which means the load level created from the workload driver is the same in all cases.
- These parameters are varied:
  - The number of guests
  - The number of virtual CPUs (VCPUs) per guest, which varies the total number of virtual CPUs in use.
  - The number of virtual CPUs per guest followed the rule that no guest should have more virtual CPUs than the z/VM has physical CPUs.
  - The distribution of the JVMs among the guests.
- The WebSphere parameters were adjusted so that no swap space is used on any guest configuration.
- The following fields were set in all the WebSphere Application Servers:

```
WebSphere configuration:
-----------------------

   Enforce Java2 Security:      false

Servers:
   server1

 EJB/ORB -----------------------------------------
   NoLocalCopies:               true
 Web ---------------------------------------------
   Min WebContainer Pool Size:  20
   Max WebContainer Pool Size:  20
 JVM ---------------------------------------------
   Min JVM Heap Size:           700
   Max JVM Heap Size:           700
   Verbose GC:                  true
   Generic JVM Arguments:

 Logging -----------------------------------------
   System Log Rollover Type:    NONE
   Trace Specification:         *=all=disabled
   Rollover Size:               100
   Max Backup Files:            10
 Misc --------------------------------------------
   Enable PMI Service:          false

   Uninstall default apps:      true
```

Test scenarios

Six test scenarios were used, two of which were for setup parameters:

Test scenario 1: Guest scaling
Scale the number of guests: 2, 4, 10, 20, 50, 100, 200, and distribute the JVMs as described in Table 4.

Table 4. Test scenario 1 - Guest scaling: Test case configurations

Number of JVMs per guest	Number of guests	Number of virtual CPUs per guest	Total number of virtual CPUs	Ratio of virtual to real CPUs	Number of JVMs per virtual CPU	Guest memory size in GB	Total virtual memory size in GB	Comments
200	1	24	24	1.0:1	8.3	200	200	
100	2	12	24	1.0:1	8.3	100	200	

Number of JVMs per guest	Number of guests	Number of virtual CPUs per guest	Total number of virtual CPUs	Ratio of virtual to real CPUs	Number of JVMs per virtual CPU	Guest memory size in GB	Total virtual memory size in GB	Comments
50	4	6	24	1.0:1	8.3	50	200	
20	10	3	30	1.3:1	6.7	20	200	CPU over commitment
10	20	2	40	1.7:1	5.0	10	200	
4	50	1	50	2.1:1	4.0	4	200	Uniprocessor and CPU over commitment
2	100	1	100	4.2:1	2.0	2	200	
1	200	1	200	8.3:1	1.0	1	200	

The results of this test can be found in Test scenario 1: Guest scaling.

Test scenario 2: Varying the number of virtual CPUs for 20 guests

A CPU scaling was done for the 20 guest scenario with 10 JVM per guest.

The results of this test can be found in Test scenario 2: Varying the number of virtual CPUs for 20 guests.

Test scenario 3: Varying the number of virtual CPUs for 200 guests

A CPU scaling was also done for the 200 guest scenario with one JVM per guest.

The results of this test can be found in Test scenario 3: Varying the number of virtual CPUs for 200 guests.

Test scenario 4: Virtual CPU scaling and WebSphere threads

The number of virtual CPUs was varied to values of: 1, 2, 3, and 24.

The results of this test can be found in Test scenario 4: Virtual CPU scaling and WebSphere threads.

Setup test 1: Large guests

Define a single guest with 200 WebSphere Application Server nodes.

The results of this test can be found in Setup test 1: Large guests.

Setup test 2: Using a shared minidisk for WebSphere binaries with and without MDC

Replace the DCSS within the WAS installation tree by a shared read-only disk, enabled for MiniDiskCache.

Repeat the test without MiniDiskCache.

The results of this test can be found in Setup test 2: Using a shared minidisk for WebSphere binaries with and without MDC.

Measurement results

This section presents the results of four tests that compare the effects of WebSphere Application Server vertical JVM stacking to horizontal JVM stacking, when altering various parameters.

- Test scenario 1: Guest scaling
- Test scenario 2: Varying the number of virtual CPUs for 20 guests
- Test scenario 3: Varying the number of virtual CPUs for 200 guests
- Test scenario 4: Virtual CPU scaling and WebSphere threads

Two setup-related tests were also performed. They are described in Setup tests.

Test scenario 1: Guest scaling

Guest scaling was done to observe the impact on CPU utilization.

Figure 2 shows the throughput and CPU load as the number of guests is scaled from 1 to 200, according to Table 4. The total number of JVMs is kept at a constant value of 200. The constant value is maintained by decreasing the number of JVMs per guest as the number of guests goes up.

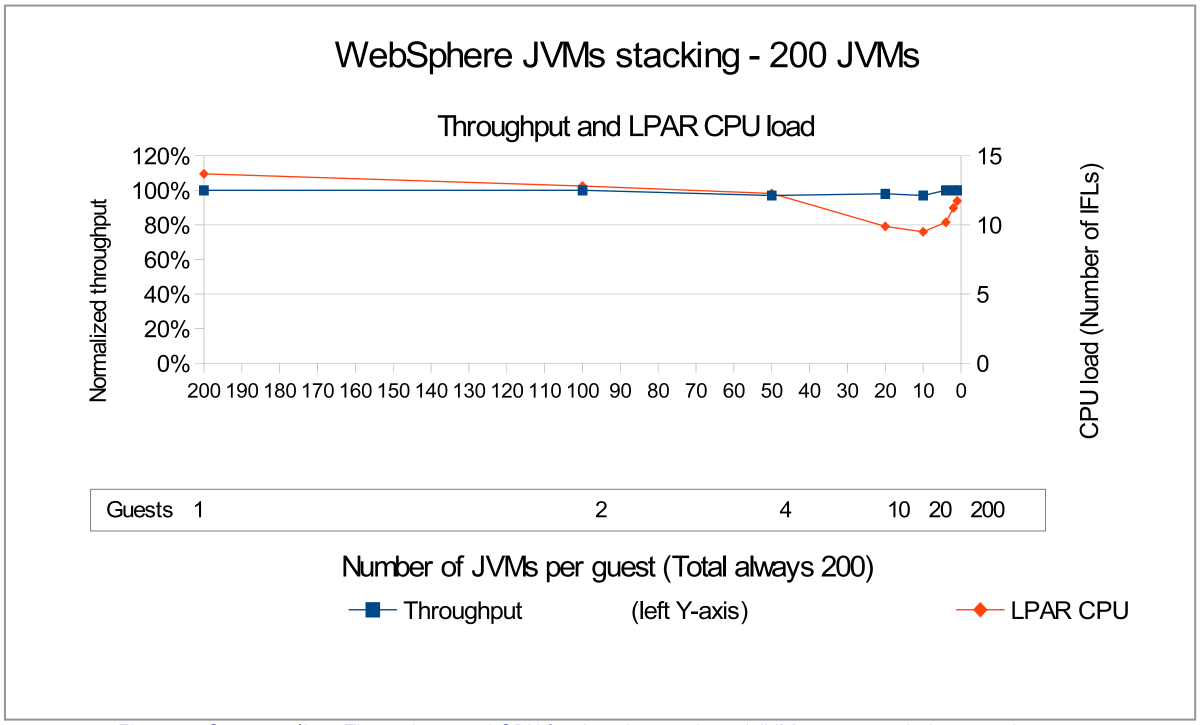


Figure 2. Guest scaling: Throughput and CPU load as the number of JVMs per guest is increased

Observations

The impact of distributing the number of JVMs (WebSphere Application Servers) across z/VM guests has a moderate effect on throughput. There is a 3% difference between the maximum throughput and the minimum throughput. One JVM per guest (200 guests) has the highest throughput. The scenario with 10 guests and 20 JVMs per guest has the lowest throughput.

- With 10 guests and 20 JVMs per guest, we start to observe CPU overcommitment. The CPU overcommitment continues up to the 200 guests with one JVM per guest test.

- The scenarios with 50 guests (four JVMs per guest) and more are Linux Uniprocessor environments. These environments do not have the Linux overhead related to multiprocessor environments, for example there are no spin locks.
- The environment was defined to never have memory overcommitment.

The most important impact is in the LPAR CPU load. The effect of stacking JVMs on fewer guests is significant. The difference between the maximum load (13.7 IFLs) with 200 JVMs per guest and the minimum load (8.8 IFLs) with 10 JVMs per guest is nearly 4 IFLs. A setup with 20 or more JVMs per guest could be considered untypical. For more common configurations, with 1 - 20 JVMs per guest, the variation is approximately 19%. There is a difference of 2 IFLs between the minimum and the maximum.

Conclusion
In the workload used in these tests, there appears to be little difference in throughput between vertical WebSphere JVM stacking and horizontal WebSphere JVM stacking.

However, LPAR CPU utilization is a different story. Some configurations seem to consume considerably fewer CPU resources than others. The next step is to identify the cause of that variation.

z/VM CPU load

To understand better what causes this large difference in CPU load, the following two figures show the behavior of the z/VM CPU load as the number of guests are scaled from 1 to 200.

As the z/VM Control Program runs, it records how it spends its time on each of its logical CPUs. z/VM accrues the time for each logical CPU into four categories:

1. Time that the virtual CPU spends running guests, referred to as guest time, virtual time, or emulation time. All three terms have the same meaning.
2. Time that the virtual CPU spends running the Control Program, doing work in support of a specific action taken by a specific guest. An example is to drive a virtual network interface. Referred to as CP time.
3. Time that the virtual CPU spends running the Control Program, doing system management functions. These functions are not attributable to the direct actions of any guest, and therefore are not chargeable to any guest. Referred to as system time.

The values used in this paper were obtained from the z/VM Perfkit report FCX144 PROCLOG.

This report shows the logical CPUs utilization with the four z/VM time categories combined as follows:

1. The ***Emulation*** time is time spent running guests, assigned to category 1 above.
2. The ***User*** time is the sum of categories 1 and 2 above. In other words, time either used directly by guests or directly chargeable to them as Control Program overhead that they caused.
3. The ***System*** time is category 3, nonchargeable Control Program overhead.

Figure 3 shows the amount of CPU spent for Emulation, CP, and System when scaling the number of JVMs per guest.

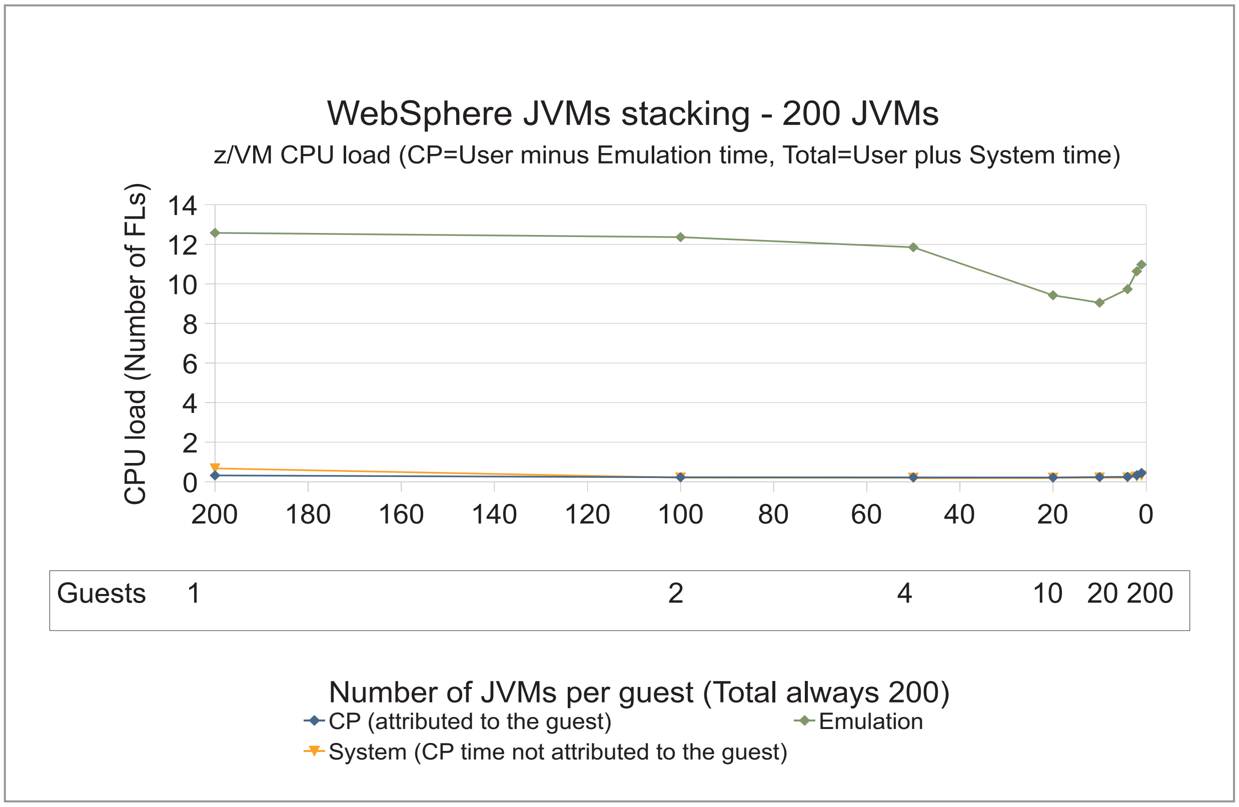


Figure 3. Amount of CPU spent for Emulation (guest load), CP (effort attributed to the guest), and System (CP overhead) in z/VM, when scaling the number of JVMs per guest

Observation
The Emulation load (Linux guest load) is the major component of the total load. This component causes the major contribution in the variation in CPU load. The variation for CP and System CPU is very moderate.

Conclusion
The variation in CPU load is not caused by virtualization overhead. The reason must be a condition inside the Linux guests.

To better depict the JVM scaling impact on the effort expended from z/VM, Figure 4 shows only CP and System CPU.

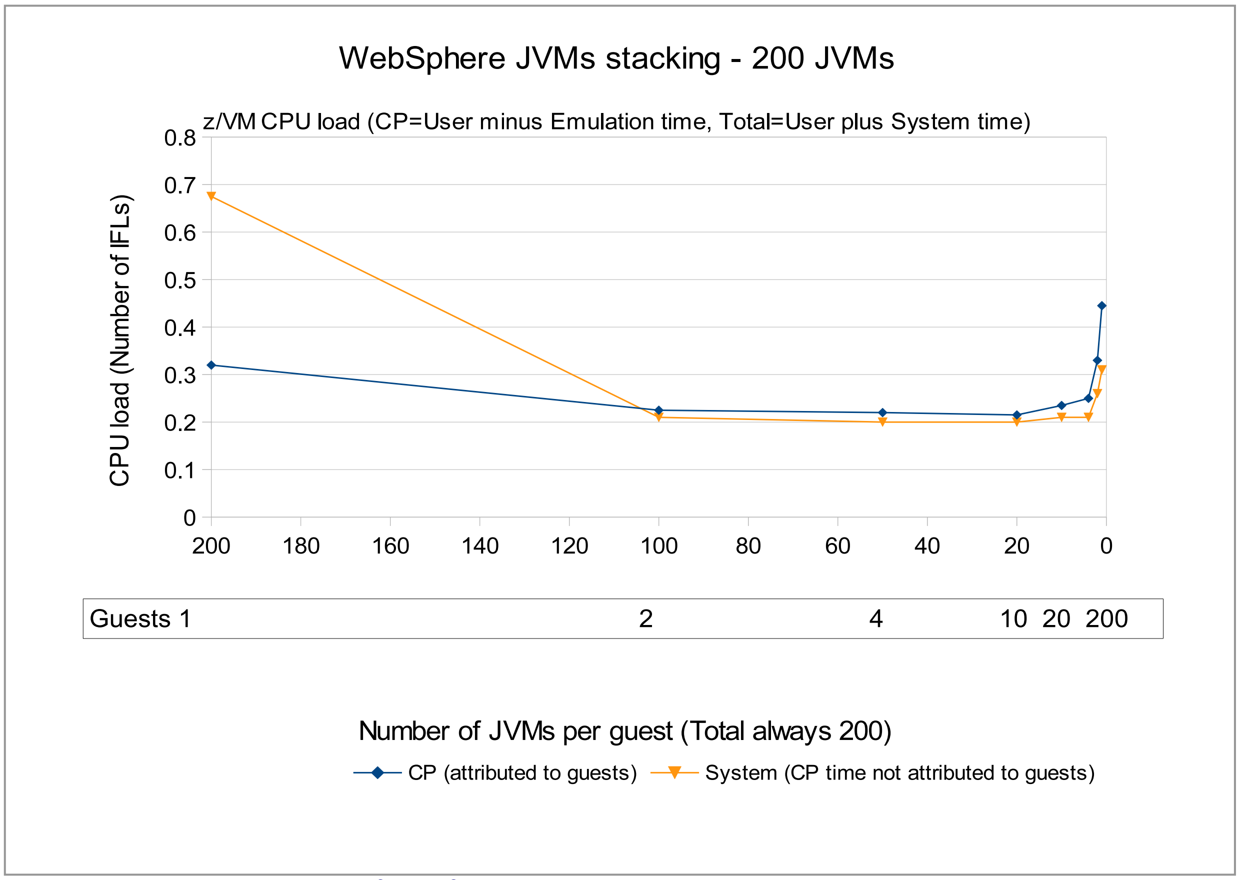


Figure 4. z/VM CP and System load when scaling the number of JVMs per guest

Observations
When scaling the number of JVMs per guest, the characteristics for z/VM management effort change. With many JVMs per guest, System time is the highest, and it is much greater than the effort CP attributes to the guest.

In a medium area, from 100 to 10 JVMs per guest, it is constant. Each part has nearly the same size and both values are lowest. When the number of guests increases by about 50, and the level of CPU overcommitment increases above two, both parameters increase heavily. But the CP time attributed to the guest becomes much higher.

However, the sum of CP effort varies from 0.4 to 1 IFL, which could be considered low when compared to the total system load.

Conclusion

This view shows the contribution of the virtualization to CPU cost. The highest efforts are related to the extreme scenarios: either one very large guest that handles everything, or a high number of small guests that drive a low workload. One obvious component of the latter scenario is that z/VM has to emulate a very large number of virtual network devices (shown as CP effort attributed to the guest).

Linux CPU load

Figure 5 shows the CPU utilization from inside Linux, in order to understand where the CPU cycles are spent in Linux.

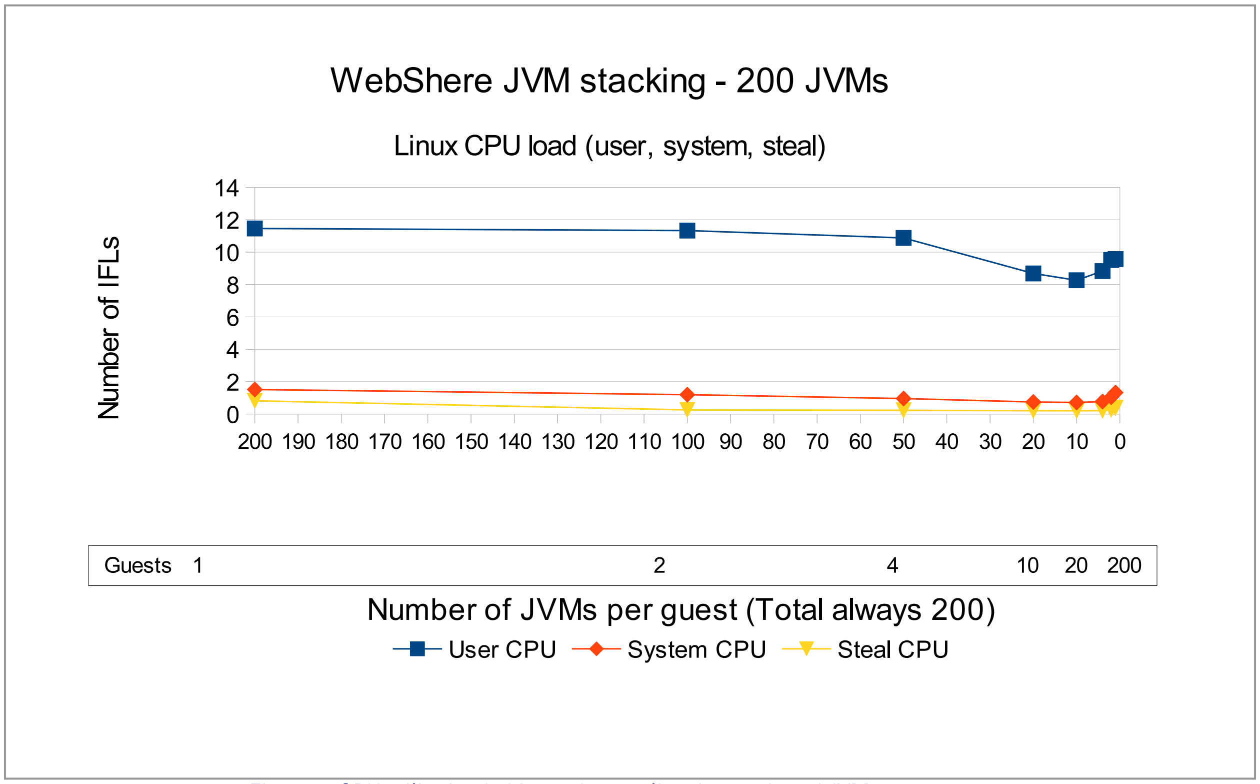


Figure 5. CPU utilization in Linux when scaling the number of JVMs per guest

Observations

The Linux user utilization is the major contributor to overall utilization. This utilization is attributed to WebSphere and Java processing. System CPU utilization decreases with the decreasing number of JVMs per system until 20 guests (10 JVMs per guest). Then, the CPU utilization starts to increase again when the number of CPUs per guest is one, with 200 JVMs.

The difference between maximum user CPU at 200 JVMs per guest and minimum user CPU at 10 JVMs per guest is 3.2 IFLs. The corresponding variation in system CPU is approximately 0.8 IFLs.

Conclusions

The major amount of CPU is spent in the user space, which means CPU cycles consumed by the JVMs. The major contribution to the large variation in CPU load comes from the user space. But the effort for the Linux system is also a contributor.

What is observed so far with regards to CPU load:

- A high number of JVMs (in the order of 50 - 200) in large guests leads to the highest CPU consumption from all the scenarios. The scenario with 200 JVMs in one guest is also related to the highest values for CP system time (not attributed to guests) and Linux system time. But in all scenarios, the major contribution is the CPU consumed by the Linux user space, which is the CPU consumed by the JVMs themselves.
- The scenario with a very low number of JVMs per guest (1 or 2) and many guests was only slightly better. Here, the z/VM CP effort attributed to the guests is the highest compared to the other scenarios. This effort is caused by the large number of guests managed, and corresponds to the increasing amount of virtual network interfaces to the VSWITCHs.
- The total amount of Linux system time is also increasing. This increase relates to the fact that the reduction in system CPU per guest does not scale to the same extent as the number of systems increases. For example, the system CPU load per guest in the 200 guest scenario is only 36% less than with 100 guests, and not 50% less. The CPU time spent on the JVMs themselves is still the largest part, but it is lower at the other end of the scale, with many JVMs in large guests.
- For our scenarios, the best setup observed was the scenario with 10 JVMs per guest and 20 guests. This scenario shows the lowest CPU utilization in all categories at throughput very close to the maximum.
- Running with overcommitted CPUs by itself has a lower impact. The ideal solution had a virtual to physical CPUs ratio of 1.7:1, which is also overcommitted, and it consumes less CPU than the scenario with a 1.2:1 ratio. There are no idling systems.
- More critical seem to be the uniprocessor scenarios. In these test cases, the system CPU in Linux increases heavily.

Test scenario 2: Varying the number of virtual CPUs for 20 guests

To analyze the impact of the number of virtual CPUs per guest in more detail, the number of CPUs in the most favorite scenario with 10 JVMs per guest and 20 guests was scaled. Originally it ran with two virtual CPUs.

Figure 6 shows the effects of running the workload with 20 guests, and scaling the number of virtual CPUs per guest.

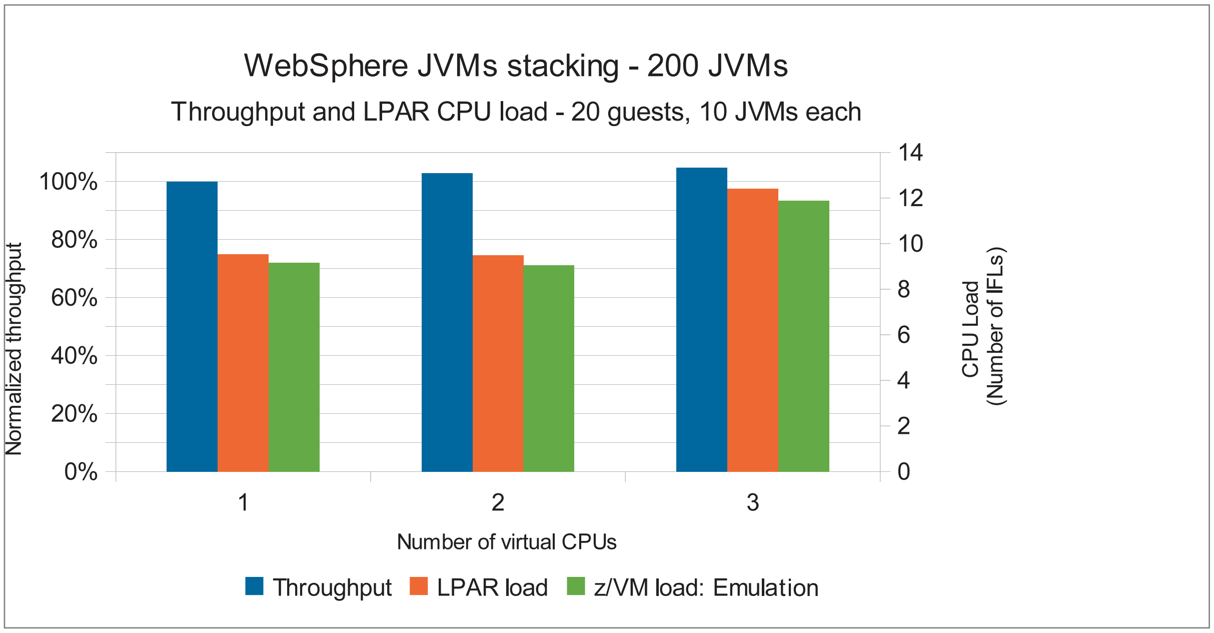


Figure 6. Throughput and LPAR CPU load - 20 guests (Emulation = guest load)

Observations

Varying the number of virtual CPUs of the guests (level of CPU overcommitment) at the lowest observed LPAR CPU load (20 guests, 10 JVMs per guest) shows a moderate effect on throughput. Throughput at one virtual CPU is the lowest, and at three virtual CPUs throughput is the highest. The difference between the minimum and maximum is 5%. The impact on LPAR CPU load is significant. Two virtual CPUs has the lowest CPU utilization, and three virtual CPUs has the highest. The difference between the minimum and maximum is 3.6 IFLs. CP effort is nearly constant; the variation is caused by the emulation part of the CPU load.

Comparing the 20 guest load at three virtual CPUs per guest against the 10 guest load at three virtual CPUs per guests shows that the 20 guest load is approximately 1 IFL higher.

Conclusions

It seems that in this scenario, the WebSphere Application Server runs better with two virtual CPUs than with one. The difference between LPAR load and Emulation (CP consumed by the guest) represents the effort expended by the z/VM CP load module. This value is fairly constant, indicating that the reason for the variation in CPU load is not z/VM overhead, but the JVMs. The load of a single guest is less than 1 IFL.

Test scenario 3: Varying the number of virtual CPUs for 200 guests

A CPU scaling was also done for the 200 guest scenario with one JVM per guest. This test was done to verify that the recommendation for running WebSphere with two virtual CPUs is valid in extreme scenarios, where the level of CPU overcommitment is very high.

Figure 7 show the effects of running the workload with 200 guests, but changing the number of virtual CPUs from 1 to 2 per guest.

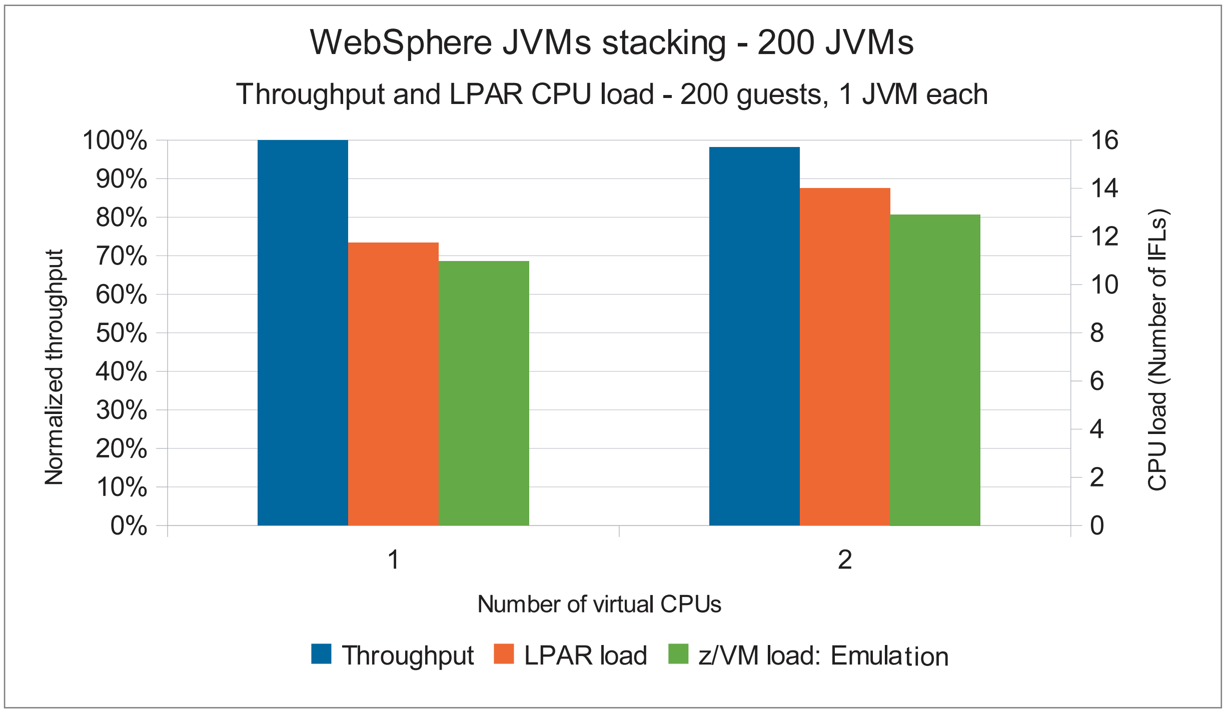


Figure 7. Throughput and LPAR CPU load - 200 guests (Emulation = guest load)

Observations

With 200 guests, the two virtual CPU case has a slightly lower throughput and a considerably higher LPAR CPU load. The LPAR CPU load increases by 2.3 IFLs, the emulation part increases only by 1.9 IFLs

Conclusions

The major contribution of the increase in CPU load still comes from the Linux guest (Emulation). But the increase of the level of CPU overcommitment from 8.3:1 to 16.7:1 causes an expected increase in z/VM effort to manage 400 virtual CPUs on 24 real CPUs. It is impressive to see that z/VM is able to handle such an excessively large number of virtual CPUs with such a low impact on total throughput.

This test also shows that the earlier finding, that the WebSphere guest runs better with two CPUs than with one for this workload, is no longer valid with these high levels of CPU overcommitment.

Test scenario 4: Virtual CPU scaling and WebSphere threads

This test varies the number of virtual CPUs, and records the number of WebSphere Application Server threads and CPU time used by these threads.

Figure 8 graphs the CPU time spent as reported from the Linux **ps -eLf** command. The times are reported in whole seconds, which can lead to a high degree of error when using many threads.

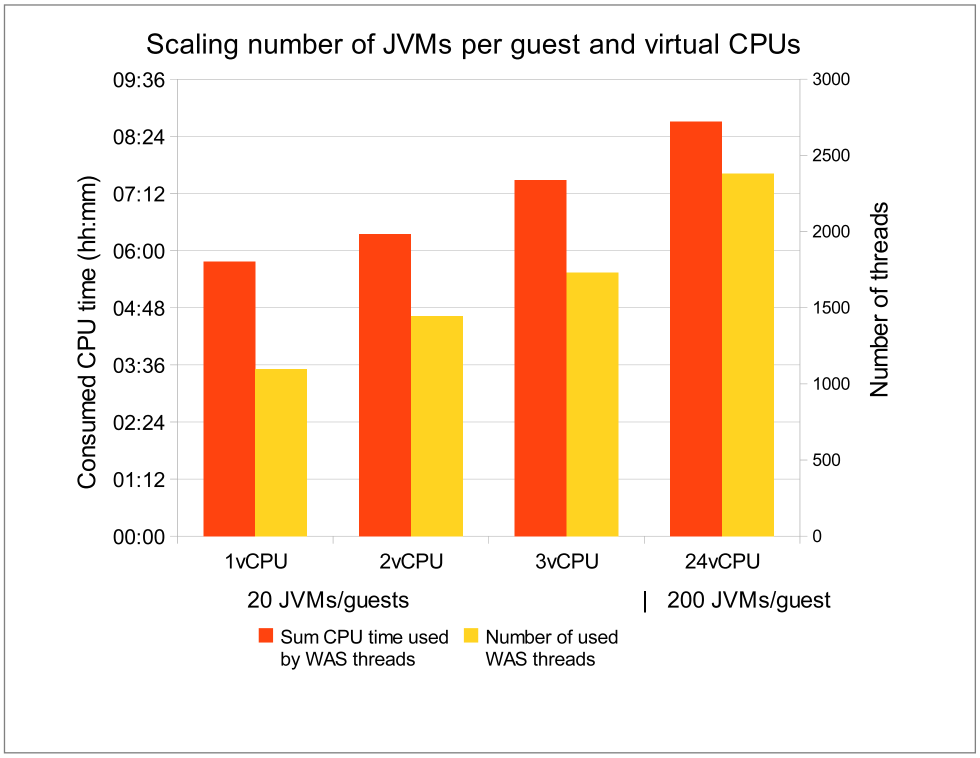


Figure 8. Scaling the number of JVMs per guest: CPU time used by WebSphere versus threads

Observations
The total number of threads used becomes higher as the number of virtual CPUs increases. The number of threads used at 20 guest with three virtual CPUs is 60% higher than with one virtual CPU. The number of threads used at one guest and 24 virtual CPUs is 120% higher. The number of CPU seconds consumed increases also, but more slowly. The scenario with three virtual CPUs consumed 30% more CPU time in total, and the scenario with 24 virtual CPUs consumed 51% more.

Conclusions
There seems to be a correlation between the number of CPUs and the number of threads *utilizing* CPU resources. With more CPUs, more threads are active.

However, the throughput across these points is similar. Part of this increase is probably due to polling.

Be aware that these values for CPU consumption time are correlated with a significant variation attributed to rounding to the nearest whole second for each of approximately 20,000 threads.

Recommendation
Do not assign more virtual CPUs to a WebSphere guest than are required. In the test environment, it was observed that even the load from a single WebSphere server is lower than 1 CPU. Two virtual CPUs for each of the 20 guests provided the best results.

Setup tests

This section presents the results of two setup tests. The first test uses a large single z/VM guest. The second test uses a shared minidisk for WebSphere binaries, with and without MDC.

Setup test 1: Large guests

The test with a large single guest revealed an interesting observation when looking at the CPU utilization. There are downward spikes at regular intervals. The single guest has a defined virtual memory size of 200 GB.

In z/VM, for each virtual machine there is a list of frames of real memory that are associated with that virtual machine. This list (*UFO List* - User Frame Owned List) is structured to facilitate finding frames of memory with different characteristics. There are two functions in z/VM to manage that list, *Demand Scan* and *Page Reorder* (or simply *Reorder*).

Demand Scan is most commonly started when z/VM detects that it has insufficient frames on its available lists. Reorder runs regardless of the status of the available list. Reorder also has a changing threshold that is partly based on the DASD paging rate. As DASD paging increases, Reorder is performed more frequently. Reorder does not really reorder pages. It is more accurate to think of it as reordering a set of pointers to pages. This is the process that is used to make sure that the virtual machine frame list (UFO List) is valid.

Part of the Reorder process deals with the hardware reference bit. Therefore, the time that it takes for the Reorder process to run is dependent on the number of resident pages of a virtual machine (because each resident page is mapped to a real frame of memory). A larger virtual machine is not a problem for Reorder, unless all the pages are resident. While Reorder is running for a virtual machine, it is stopped. All virtual machines go through Reorder at one time or another.

There are a number of factors that affect how long Reorder takes to complete. A very rough rule of thumb is one second for every 8 GB of resident memory. For virtual machines with more than one virtual processor, all processors are stopped during Reorder processing. While Reorder could occur for multiple virtual machines at the same time, it would still result in serializations of the individual virtual machines being reordered. Reorder processing does not serialize the z/VM system as a whole.

For more information about the use of Page Reordering and guidance on enabling and disabling it, see, the Reorder web page: http://www.vm.**ibm.co**m/perf/tips/reorder.html

If you have questions about the use of the **SET REORDER** command for your system, contact IBM z/VM support for assistance.

The downward spikes that are observed in Figure 9 can be attributed to the Page Reorder function described above.

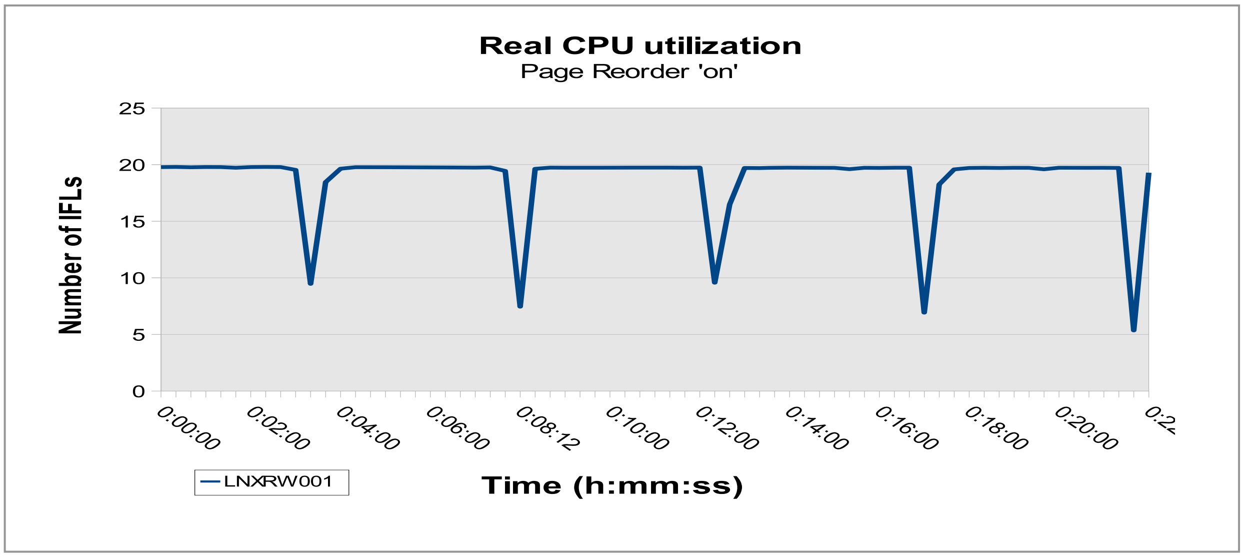


Figure 9. Real CPU utilization with Page Reorder enabled

There is a z/VM service APAR that gives the administrator the option to disable Page Reorder for a guest ID. The APAR was installed, and the Page Reorder function was set off for the guest. Figure 10 shows the CPU utilization with Page Reorder set off. There are no longer any downward spikes.

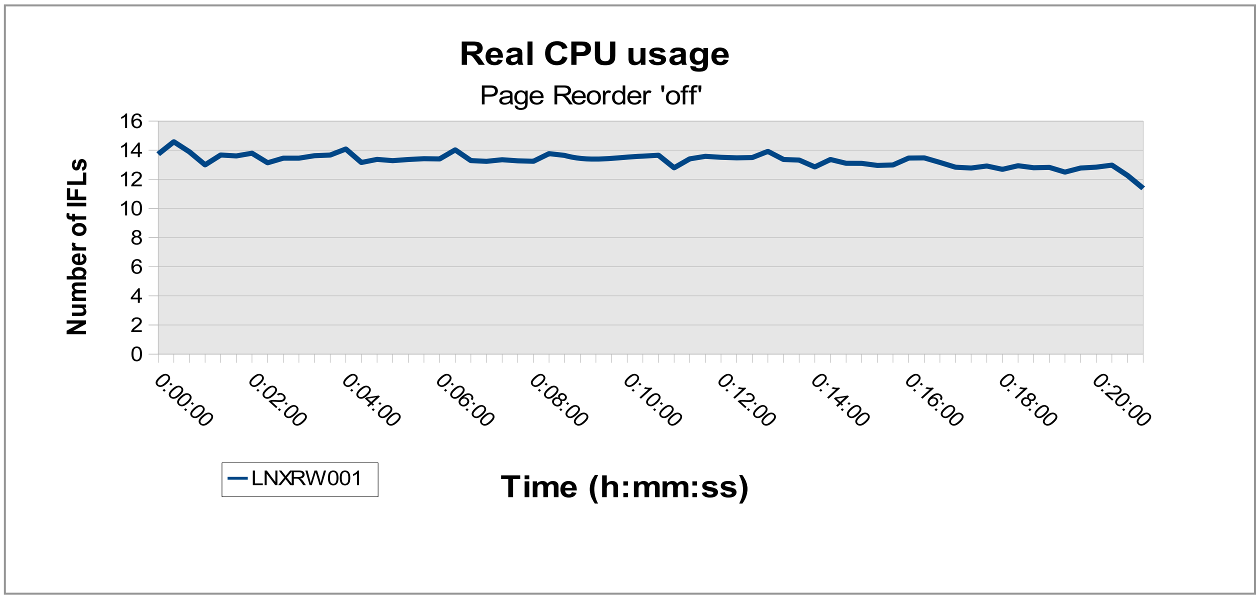


Figure 10. Real CPU utilization with Page Reorder disabled

Based on the behavior of the Page Reordering function, Page Reordering was set OFF for all tests environments where the guests virtual memory size was 8 GB or larger, which is not critical because all tests run without memory overcommitment.

Setup test 2: Using a shared minidisk for WebSphere binaries with and without MDC

This scenario replaces the DCSS with a minidisk for sharing the WebSphere binaries between guests. This test compares the minidisk environment to the DCSS environment.

Figure 11 and Figure 12 show the results for throughput and CPU utilization.

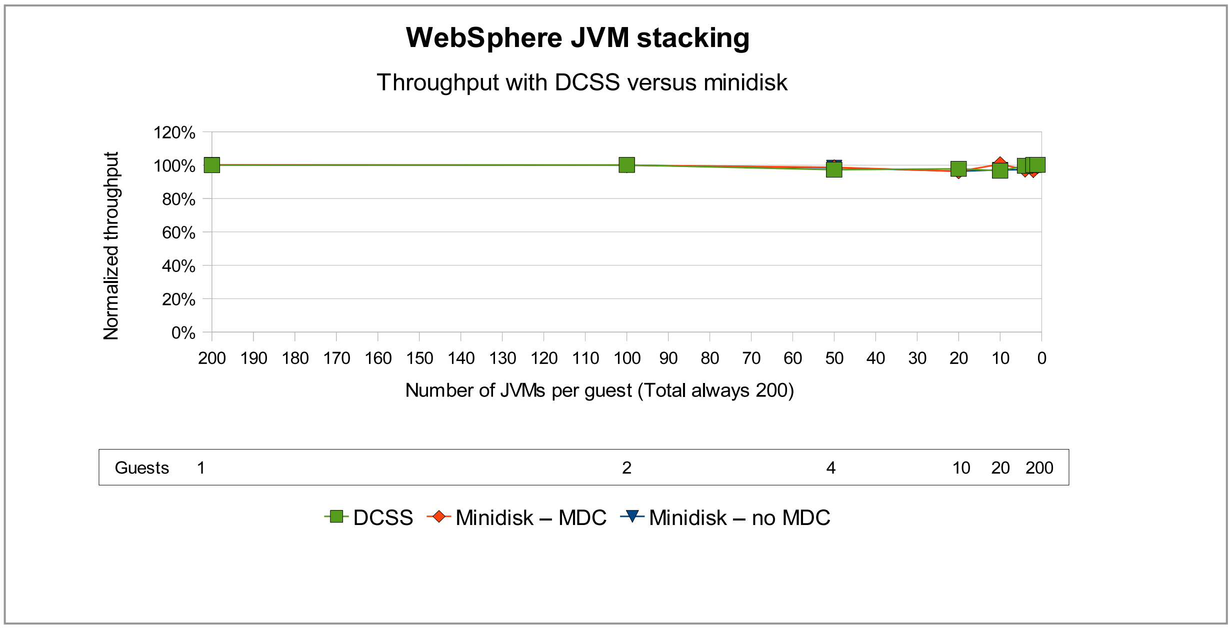


Figure 11. Throughput: DCSS versus minidisk

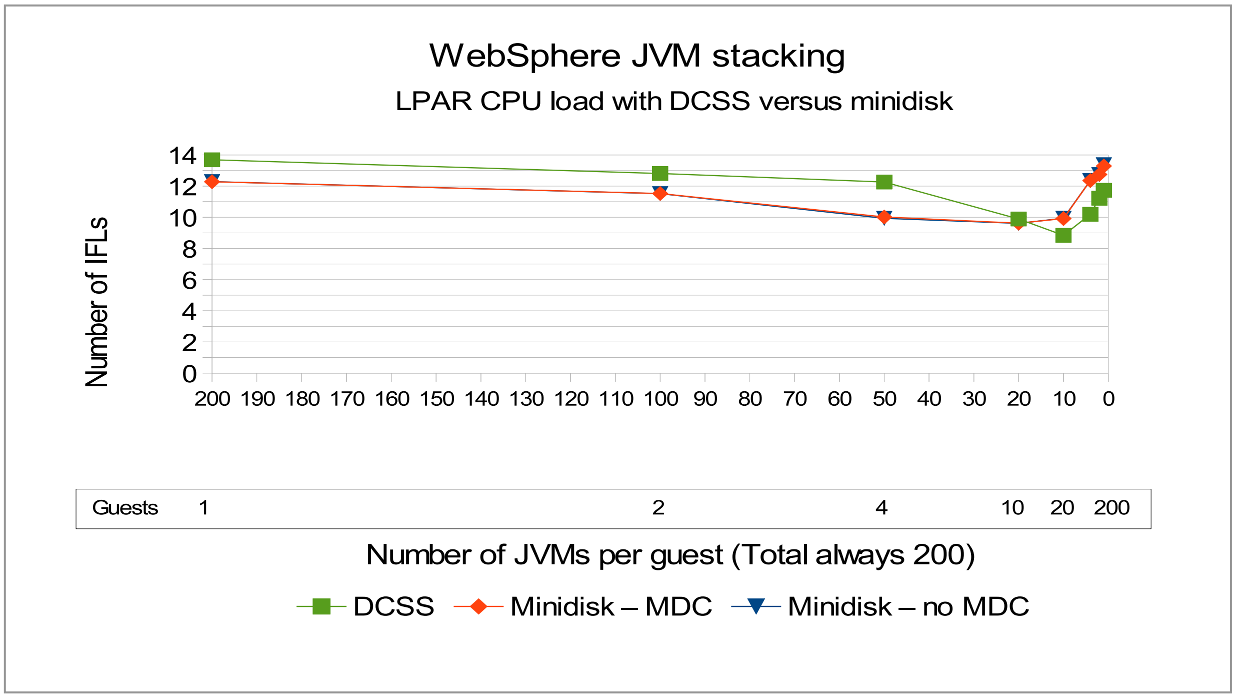


Figure 12. LPAR CPU utilization: DCSS versus minidisk

Observations
There is very little difference in throughput. The LPAR CPU utilization has a significant impact. For a small number of guests (1 - 4), CPU utilization for the minidisk case is less than the DCSS case: 1.4 IFLs compared to 2.2 IFLs. Ten guests is the break even point. With more than ten guests, the DCSS case has the lower CPU utilization: 1.5 IFLs compared to 2.2 IFLs.

Conclusion
Once again, in regard to throughput all these changes have a very low impact. But there is a significant impact in regard to CPU utilization. The additional effort related to using DCSS makes it only attractive when the amount of guests using it exceeds a certain number. With less than 10 guests, it is more efficient using a shared disk. The scenarios with more than 10 guests has a significant reduction in CPU utilization from using a DCSS.

There appears to be no advantage to minidisk caching, however it also is not a disadvantage. This result seemed to be caused by the caching in Linux in the page cache. The MDC might become more important if Linux runs with memory constraint.

Reference

These references provide details about the test environment and test setup.

- Overview of SOABench
  http://w3.tap.ibm.com/w3ki/display/SOAB/SOABench+-+The+IBM+SOA+Benchmark
- z/VM and Linux on IBM System z The Virtualization Cookbook for SLES 10 SP2
  www.redbooks.ibm.com/redbooks/pdfs/sg247493.pdf
- How to - Share a WebSphere Application Server V7 installation among many Linux for IBM System z systems
  http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=SA&subtype=WH&htmlfid=ZSW03055USEN
- A study to three areas of application for a large DCSS: sharing code, sharing read-only data, and using a DCSS as a swap device
  http://www.ibm.com/developerworks/linux/linux390/perf/tuning_vm.html#dcss
- Sharing and Maintaining SLES 11 Linux under z/VM using DCSSs and an NSS
  www.vm.ibm.com/linux/dcss/ror-s11.pdf
- For more information about the use of a DCSS under z/VM
  http://www.vm.ibm.com/linux/dcss/
- For more information to z/VM and page reorder
  http://www.vm.ibm.com/perf/tips/reorder.html

ZSW03213-USEN-00