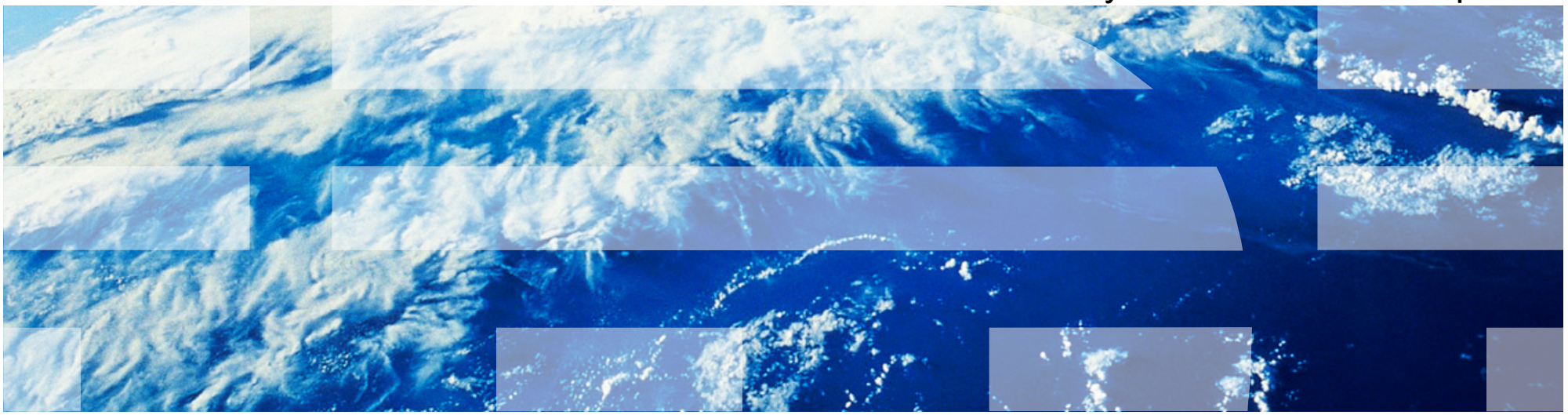
2012-03-15

Linux on System z

Optimizing Resource Utilization for Linux under z/VM – Part II

Dr. Juergen Doelle
IBM Germany Research & Development



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

Trademarks

Agenda

- Introduction
- Methods to pause a z/VM guest
- WebSphere Application Server JVM stacking
- Guest scaling and DCSS usage
- Summary

Introduction

- **A virtualized environment**
  - runs operating systems like Linux on virtual hardware
  - backing up the virtual hardware with physical hardware *as required* is the responsibility of a Hypervisor, like z/VM.
  - provides significant advantages in regard of management and resource utilization
  - running virtualized on shared resources introduces a new dynamic into the system
- **The important part is the '*as required*'**
  - normally do not all guests need all assigned resources all the time
  - It must be possible for the Hypervisor to identify resources which are not used.
- **This presentation analyzes two issues**
  - Idling applications which look so frequently for work that they appear active to the Hypervisor
    - Concerns many applications running with multiple processes/threads
    - Sample: 'noisy' WebSphere Application Server
    - Solution: pausing guests which are know to be unused for a longer period
  - A configuration question, what is better vertical or horizontal application stacking
    - stacking many applications/middle ware in one very large guest
    - having many guests with one guest for each application/middle ware
    - or something in between
    - Sample: 200 WebSphere JVMs
    - Solution: Analyze the behavior of setup variations

Agenda

- Introduction
- **Methods to pause a z/VM guest**
- WebSphere Application Server JVM stacking
- Guest scaling and DCSS usage
- Summary

Introduction

- **The requirement:**
  - An idling guest should not consume CPU or memory.
  - Definition of idle: A guest not processing any client requests is idle
    - This is the customer view, not the view from the operating system or Hypervisor.
- **The issue:**
  - An 'idling' WebSphere Application Server guest is so frequently looking for work, that it does not become idle from the Hypervisor view.
  - For z/VM: it can not set the guest state to dormant and the resources, especially the memory, are considered as actively used:
    - z/VM could use real memory pages from dormant guests easily for other active guests
    - But the memory pages from idling WebSphere guests will stay in real memory and not be paged out
    - The memory pages from an idling WebSphere Application server compete with really active guests for real memory pages
  - This behavior expected to be not specific for WebSphere Application Server
- **A solution**
  - Guests which are known as inactive, for example when the developer has finished his work, are made inactive to the z/VM by
    - using Linux suspend mechanism (hibernates the Linux)
    - using z/VM stop command (stops the virtual CPUs)
  - This should help to increase the level of memory overcommitment significantly, for example for systems hosting WAS environments for a world wide working development groups

How to pause the guest

- **Linux Suspend/Resume**
  - Setup: dedicated swap disk to hold the full guest memory
zipl.conf: resume=<swap device_node>
Optional: add a boot target with the noresume parameter as failsafe entry
  - Suspend: echo disk >/sys/power/state
  - Impact: controlled halt of Linux and its devices
  - Resume: just IPL the guest
  - more details in the device drivers book:
http://www.ibm.com/developerworks/linux/linux390/documentation_dev.html

- **z/VM CP Stop/Begin**
  - Setup: Privilege class: G
  - Stop: CP stop cpu all (might be issued from vmcp)
  - Impact: stops virtual CPUs, execution just halted,
device states might remain undefined
  - Resume: CP begin cpu all (from x3270 session, disconnect when done)
  - more details in CP command reference:
http://publib.boulder.ibm.com/infocenter/zvm/v6r1/index.jsp?topic=/com.ibm.zvm.v610.hcpb7/toc.htm

Objectives - Part 1

- **Objectives – Part 1**
  - Determine the time needed to deactivate/activate the guest
  - Show that the deactivated z/VM guest state becomes dormant
  - Use a WebSphere Application Server guest and a standalone database

Time to pause or restart the guests

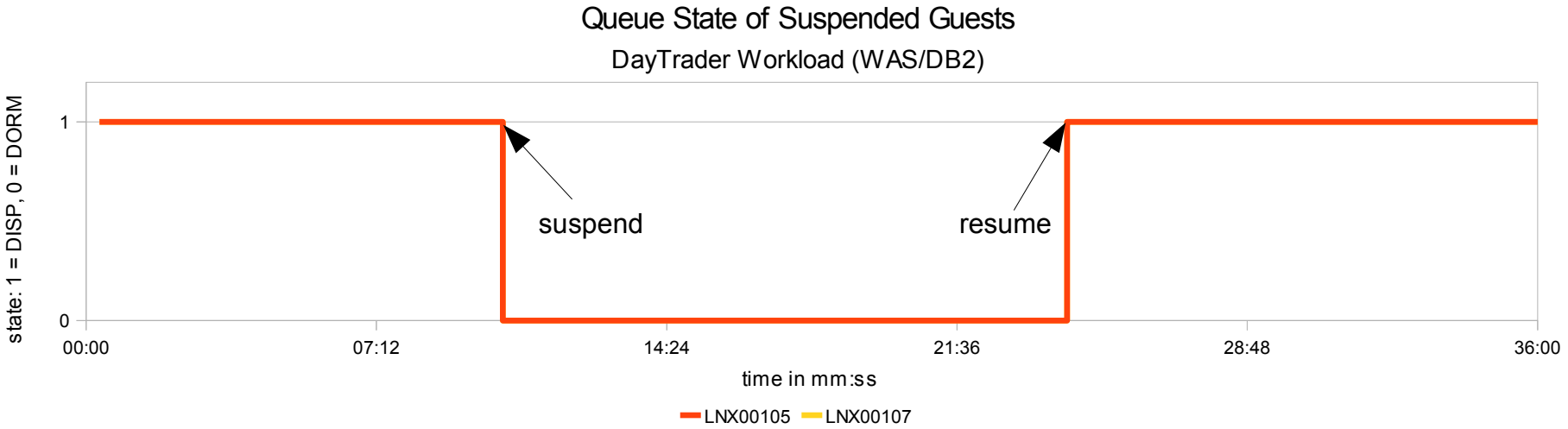
	Times until the guest is halted			
	Linux: suspend		z/VM: stop command	
	standalone database	WebSphere guest	standalone database	WebSphere guest
times [sec]	8	27	immediately	immediately

	Times until the guest is started again	
	Linux: resume	z/VM: begin command
times [sec]	19	immediately

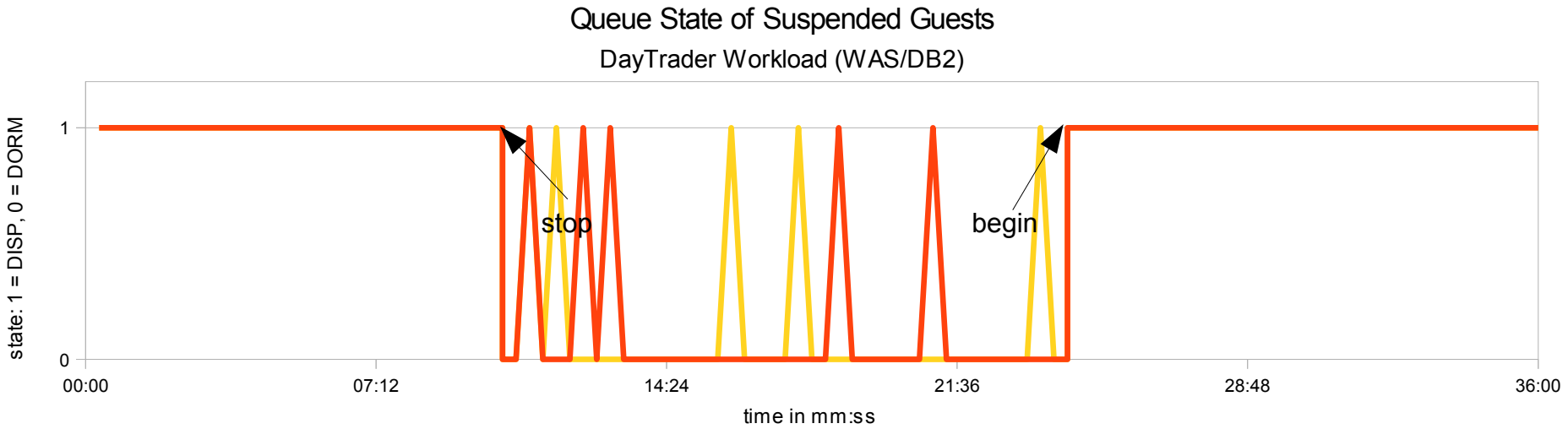
- **CP stop/begin is much faster**
  - In case of an IPL/ power loss system state is lost
  - Disk and memory state of the system might be inconsistent
- **Linux suspend writes the memory image to the swap device and shutdown the devices.**
  - Needs more time, but the system is left in a very controlled state
  - IPL or power loss have no impact on the system state
  - All devices are cleanly shutdown

Guest States

Suspend/Resume



Stop/Begin

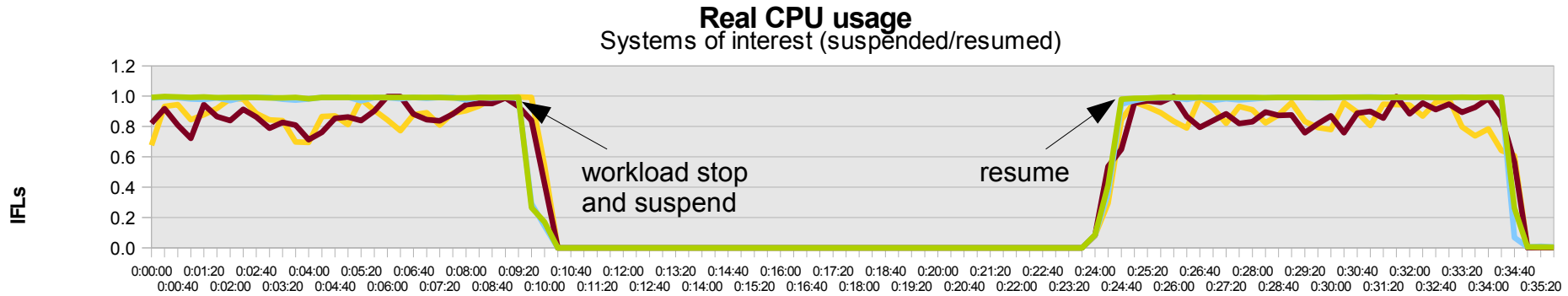


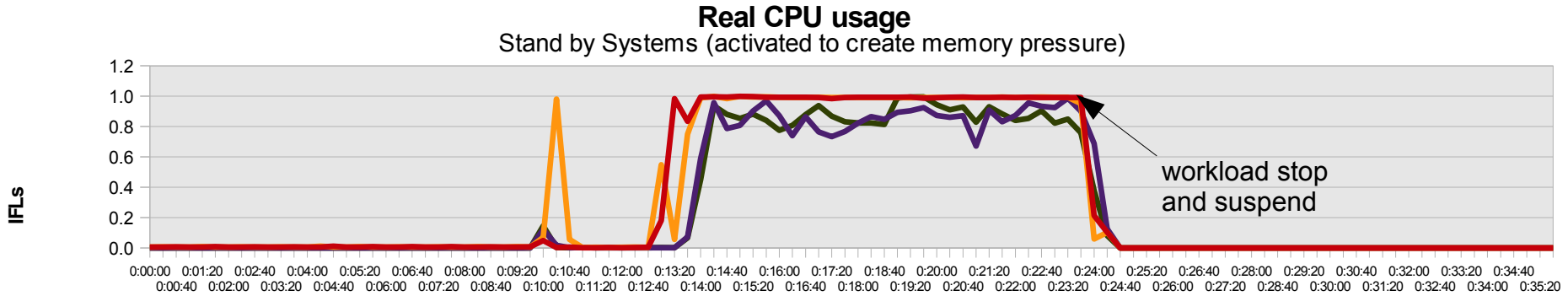
- **Suspend/Resume behaves ideal, full time dormant!**
- **With Stop/Begin the guest gets always scheduled again**
  - **on reason is that z/VM is still processing interrupts for the virtual NICs (VSWITCH)**

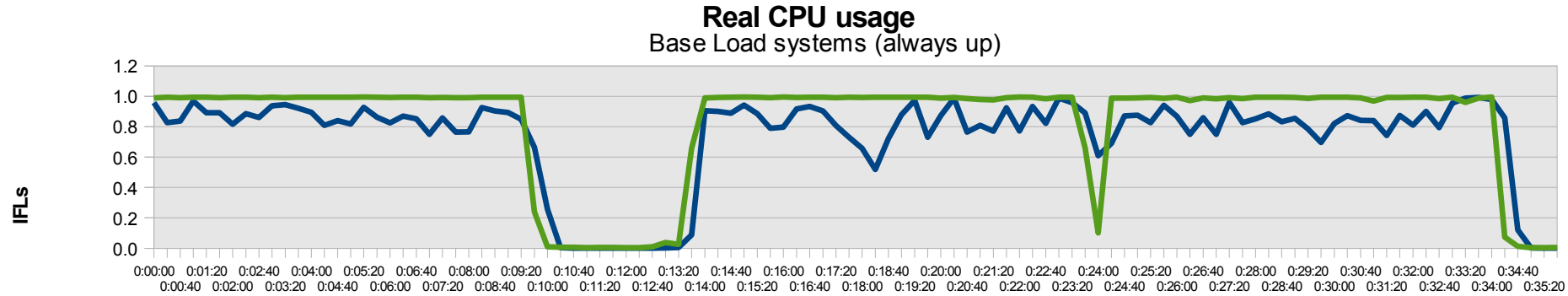
Objectives - Part 2

- **Objectives – Part 2**
  - Show if the pages from the paused guests are really moved to XSTOR
- **Environment**
  - use 5 guests with WebSphere Application Server and DB2 and 5 standalone database guests (from another vendor)
  - The application server systems are in a cluster environment which require an additional guest with network deployment manager
  - **4 Systems of interest**: target systems
    - 2 WebSphere + 2 standalone databases
  - **4 Standby systems**: activated to produce memory pressure when systems of interest are paused
    - 2 WebSphere + 2 standalone databases
  - **2 Base load systems**: never paused to produce a constant base load
    - 1 WebSphere + 1 standalone database

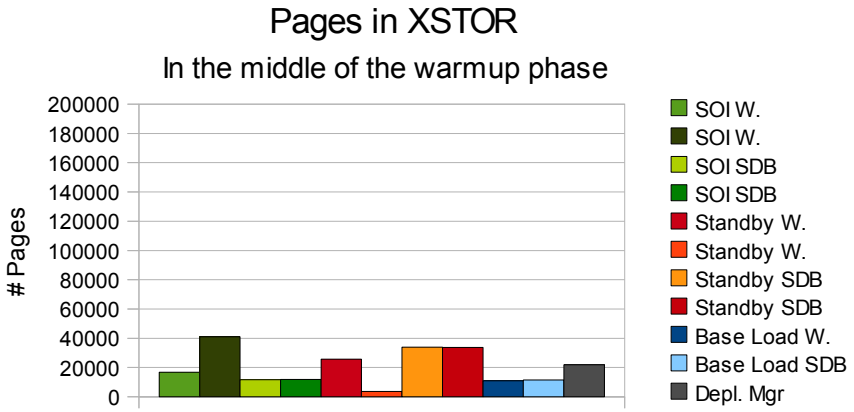
Suspend Resume Test - Methodology

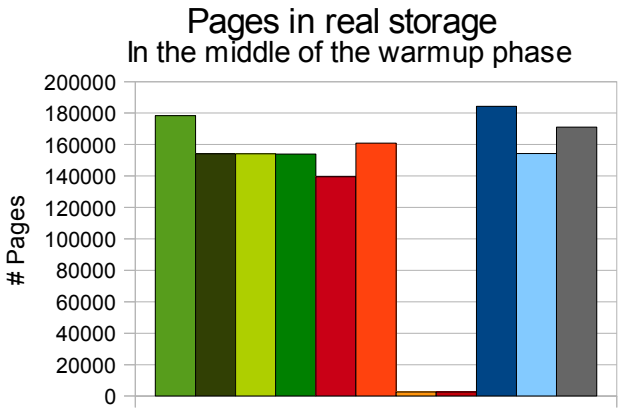


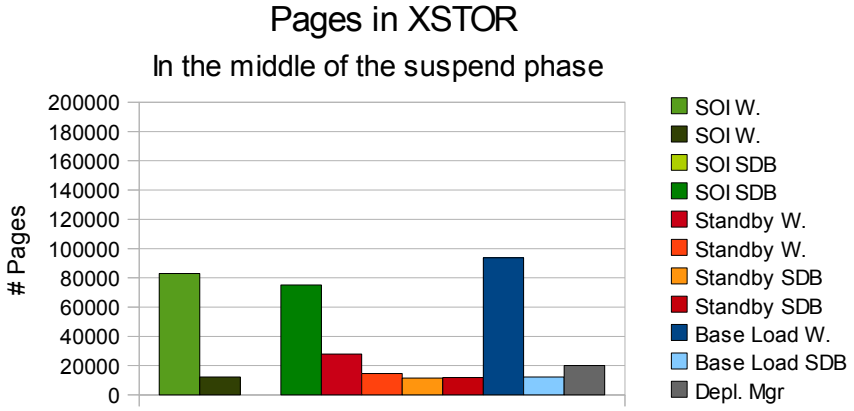


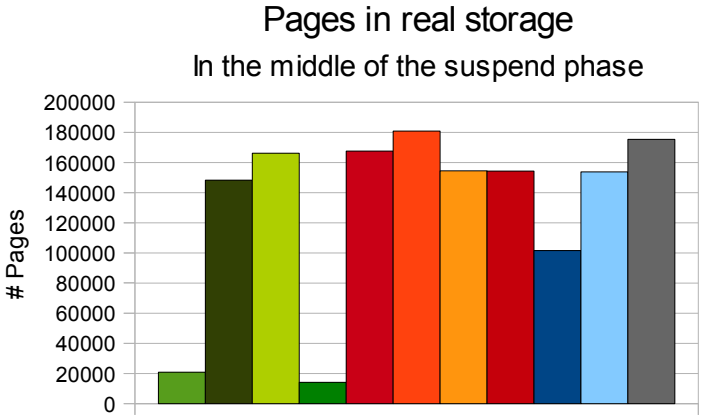


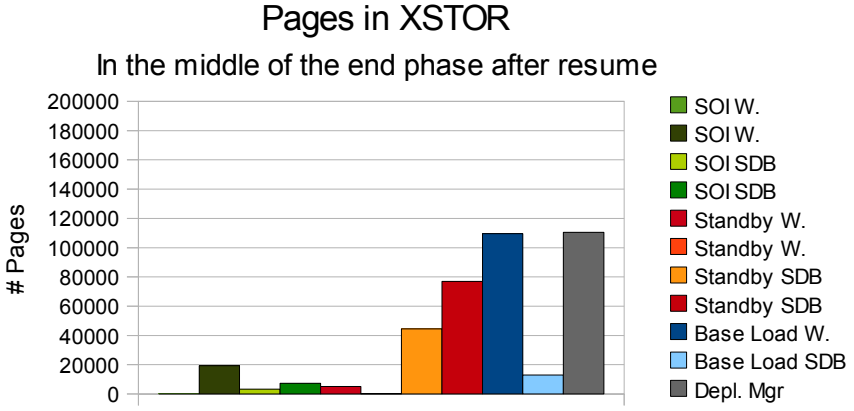
Suspend/Resume Test – What happens with the pages?

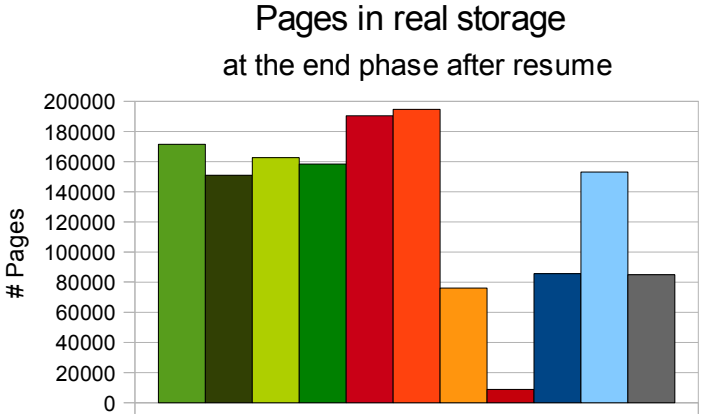




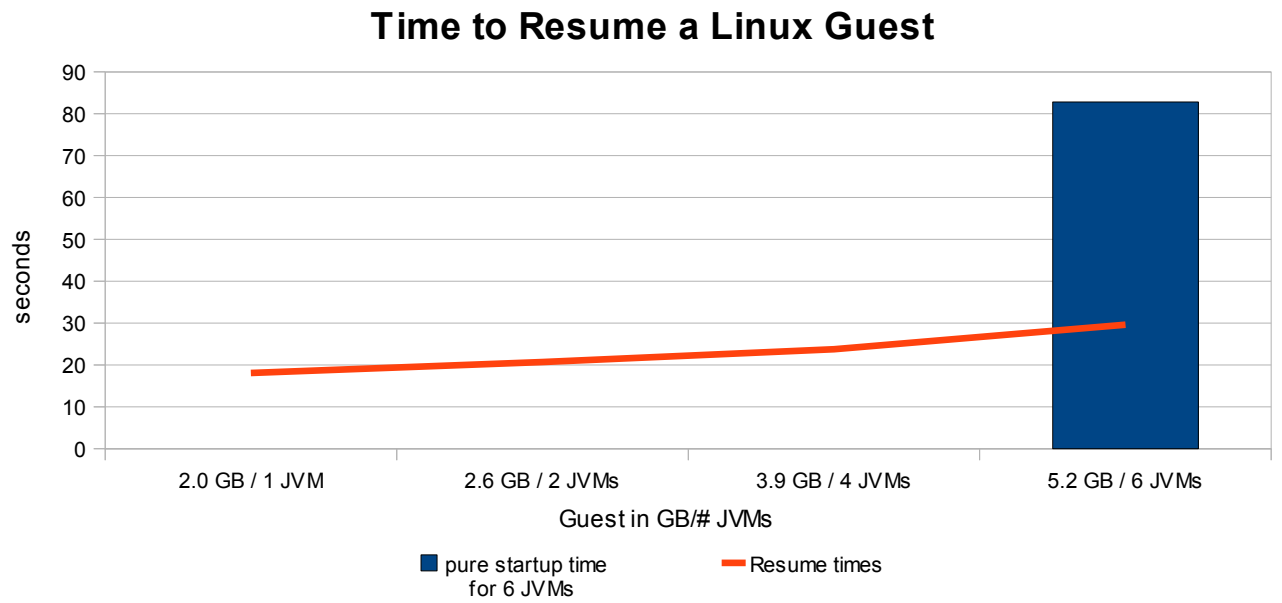








Resume Times when scaling guest size



- **The tests so far have been done with relatively small guests (<1GB memory)**
- **How does the resume time scale when the guest size increases**
  - Run multiple JVMs inside the WAS instance, each JVM had a Java heap size of 1 GB to ensure that the memory is really used.
  - With the guest size the number of JVMs was scaled
  - The size of the guest was limited by the size of a the mod9 DASD used as swap device for the memory image
- **Resume time includes the time from IPL'ing the suspended guest until successfully perform an http get**
- **Startup time includes only time for executing the startServer command for server 1 – 6 serially with a script**

▶ **Resume time was always much shorter than just starting the WebSphere application servers**

Agenda

- Introduction
- Methods to pause a z/VM guest
- **WebSphere Application Server JVM stacking**
- Guest scaling and DCSS usage
- Summary

Which setup could be recommended: one or many JVMs per guest?

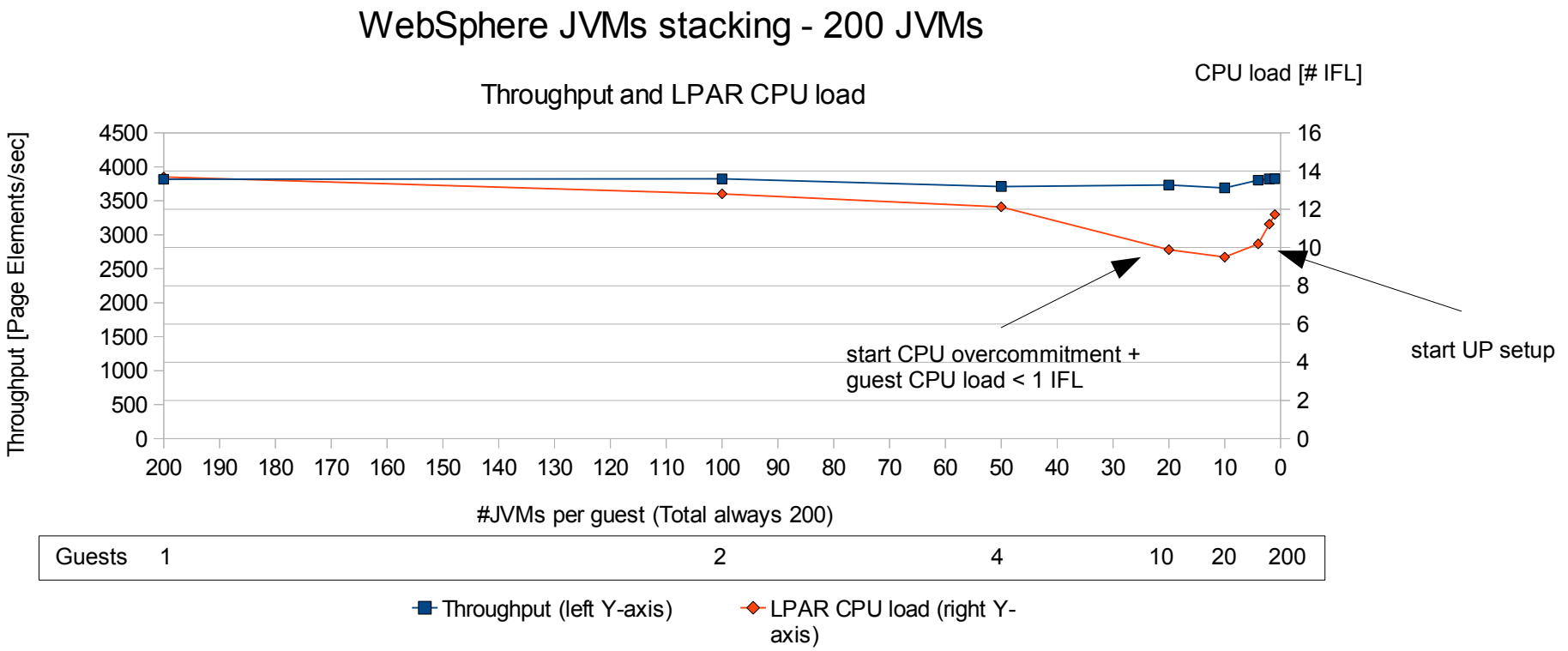
- **Expectations**
  - Many JVMs per guest are related with contention inside Linux (memory, CPUs)
  - Many guests are related with more z/VM effort and z/VM overhead (e.g. virtual NICs, VSWITCH)
  - Small guests with 1 CPU have no SMP overhead (e.g. no spinlocks)
- **Methodology**
  - Use a total of 200 JVMs (WebSphere Application Server instances/profiles)
  - Use a pure Application Server workload without database
  - Scale the amount of JVMs per guest and the amount of guest accordingly to reach a total of 200
  - Use one node agent per guest

Test Environment

- **Hardware:**
  - 1 LPAR on z196, 24 CPUs, 200GB Central storage + 2 GB expanded,
- **Software**
  - z/VM 6.1, SLES11 SP1, WAS 7.1
- **Workload: SOA based workload without database back-end (IBM internal)**
- **Guest Setup:**
  - no memory overcommitment

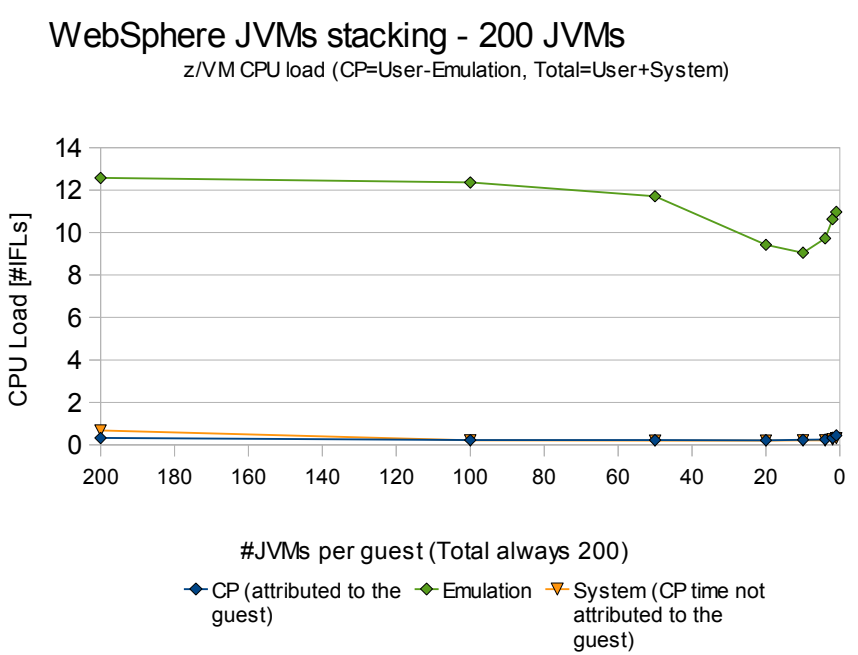
#JVMs per guest	#guests	#VCPUs per guest	total of #VCPUs	CPU real : virt	JVMs per vCPU	guest memory size [GB]	total virtual memory size [GB]	
200	1	24	24	1 : 1.0	8.3	200	200	
100	2	12	24	1 : 1.0	8.3	100	200	
50	4	6	24	1 : 1.0	8.3	50	200	
20	10	3	30	1 : 1.3	6.7	20	200	CPU Overcommitment
10	20	2	40	1 : 1.7	5.0	10	200	
4	50	1	50	1 : 2.1	4.0	4	200	Uniprocessor
2	100	1	100	1 : 4.2	2.0	2	200	
1	200	1	200	1 : 8.3	1.0	1	200	

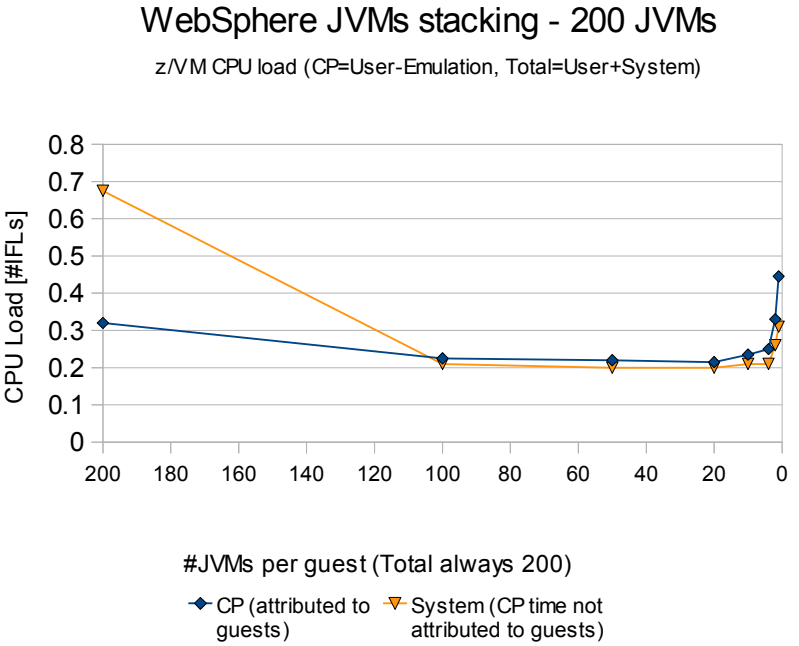
Horizontal versus Vertical WAS Guest Stacking



- **Impact of JVM stacking on throughput is moderate**
  - Minimum = Maximum – 3.5%
  - 1 JVM per guest (200 guests) has the highest throughput
  - 10 JVMs per guest (20 guests) has the lowest throughput
- **Impact of JVM stacking on CPU load is heavy**
  - Minimum = Maximum - 31%, Difference is 4.2 IFLs
  - 10 JVMs per guest (20 guests) has the lowest CPU load (9.5 IFL)
  - 200 JVM per guest (1 guest) has the highest CPU load (13.7 IFL)
- **Page reorder**
  - impact of page reorder on/off on a 4GB guest within normal variation
- **VM page reorder off**
  - for guests with 10 JVMs and more (> 8 GB memory)
- **CPU overcommitment**
  - starts with 20 JVMs per guest
  - and increases with less JVMs per guest
- **Uniprocessor (UP) setup for**
  - 4 JVMs per guest and less
- **no memory overcommmitment**

Horizontal versus Vertical WAS Guest Stacking - z/VM CPU load



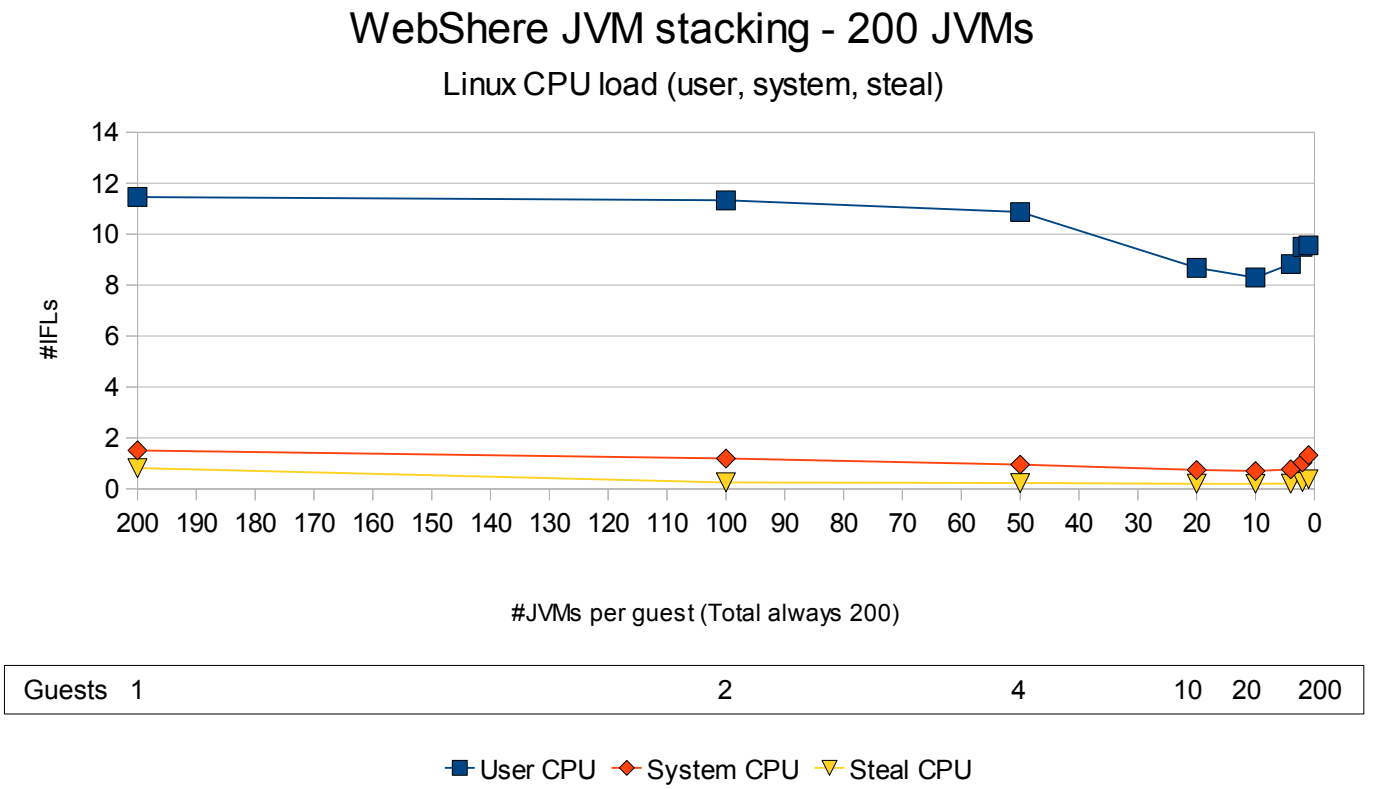


- **Which component causes the variation in CPU load?**
  - CP effort in total between 0.4 and 1 IFL
    - System related is at highest with 1 guest
    - CP effort guest related is at highest with 200 guest,
    - At lowest between 4 and 20 guests.
  - Major contribution comes from the Linux itself (Emulation).

- **z/VM CPU load consist of:**
  - Emulation → this runs the guest
  - CP effort attributed to the guest → drives virtual interfaces, e.g VNICs, etc
  - System (CP effort attributed to no guest) → pure CP effort

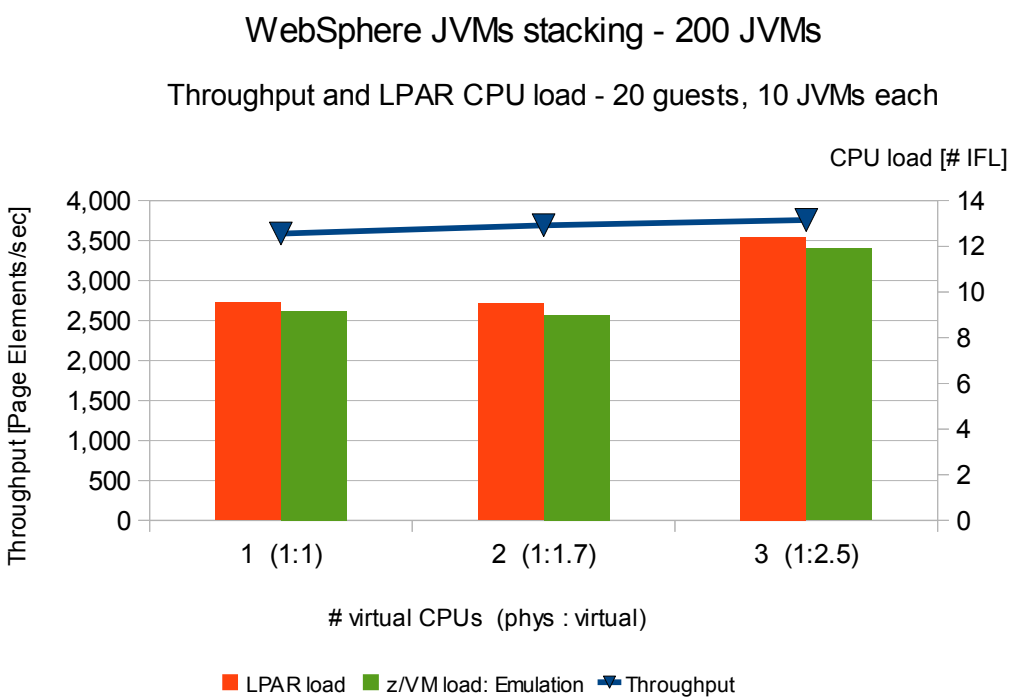
- **VM page reorder off**
  - for guests with 10 JVMs and more (> 8 GB memory)
- **CPU overcommitment**
  - starts with 20 JVMs per guest
  - and increases with less JVMs per guest
- **Uniprocessor (UP) setup for**
  - 4 JVMs per guest and less
- **no memory overcommmitment**

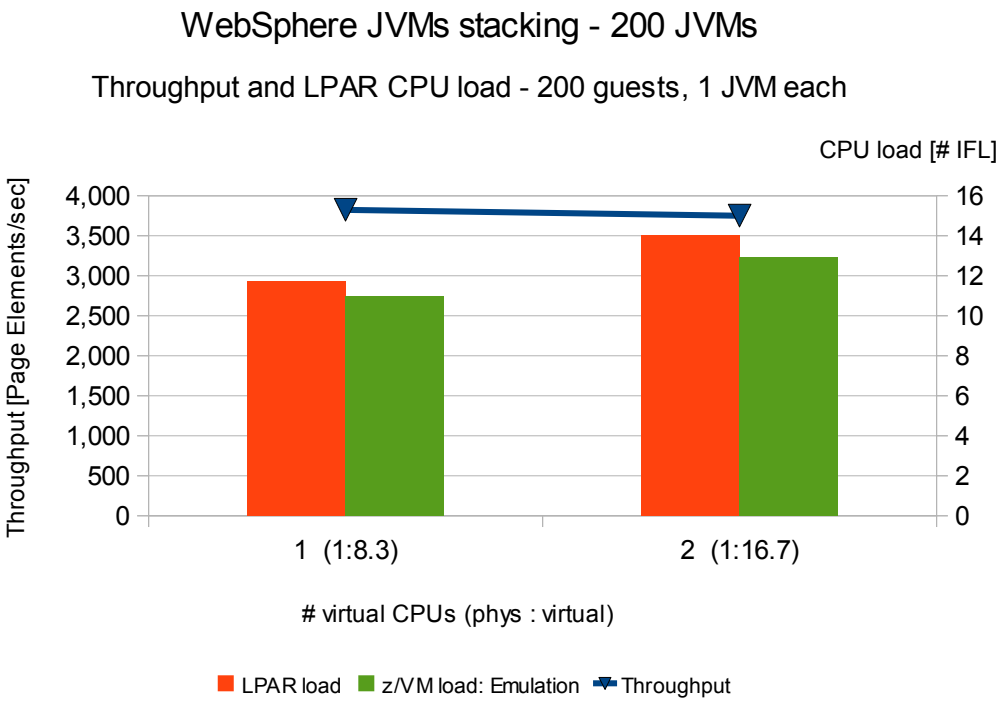
Horizontal versus Vertical WAS Guest Stacking -Linux CPU load



- **Which component inside Linux causes the variation in CPU load?**
  - Major contribution to the reduction in CPU utilization comes from the user space, e.g. inside WebSphere Application Server JVM
- **System CPU**
  - decreases with the decreasing amount of JVMs per System,
  - but increases with the Uniprocessor cases
- **The amount of CPUs per guest seems to be important!**

virtual CPU scaling with 20 guests (10 JVMs each) and with 200 guests



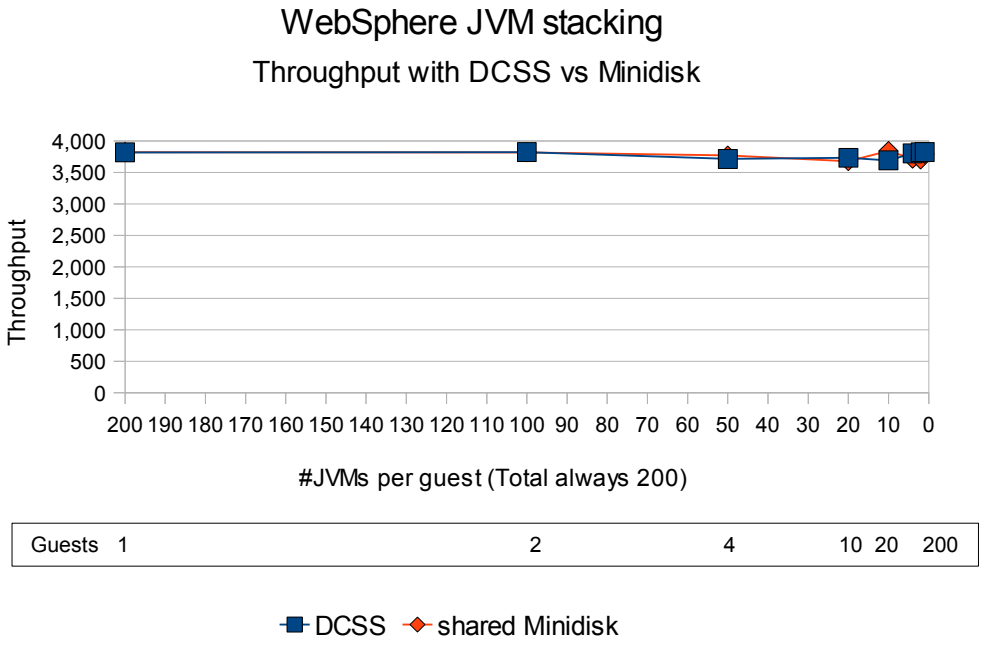


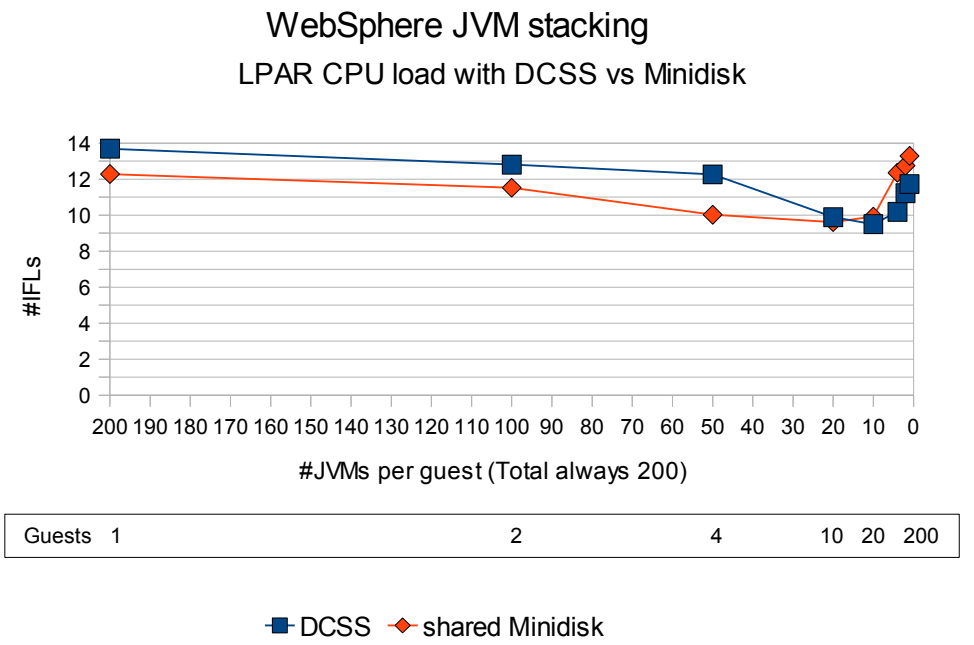
- **Scaling the amount of virtual CPUs of the guests at the point of the lowest LPAR CPU load**
  - Impact of throughput is moderate (Min = Max – 4.5%), throughput with 1 vCPU is the lowest
  - Impact on CPU load is high, 2 virtual CPUs provide the lowest amount of CPU utilization
  - The variation is caused by the emulation part of the CPU load => Linux
- **It seems that the amount of virtual CPUs has a severe impact on the total CPU load**
  - CPU overcommitment level is one factor, but not the only one!
  - WebSphere Application Server runs on Linux better with 2 virtual CPUs in this case as long as the CPU overcommitment level is not too excessive

Agenda

- Introduction
- Methods to pause a z/VM guest
- WebSphere Application Server JVM stacking
- **Guest scaling and DCSS usage**
- Summary

Horizontal versus Vertical WAS Guest Stacking – DCSS vs Minidisk





DCSS or shared minidisks?

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

Agenda

- Introduction
- Methods to pause a z/VM guest
- WebSphere Application Server JVM stacking
- Guest scaling and DCSS usage
- **Summary**

Summary – Pause a z/VM guest

- **“No work there” is not sufficient that a WebSphere guest becomes dormant**
  - probably not specific to WebSphere
- **Deactivating the guest helps z/VM to identify guest pages which can be moved out to the paging devices to use the real memory for other guest**
  - two Methods
    - Linux suspend mechanism (hibernates the Linux)
    - z/VM stop command (stops the virtual CPUs)
  - Allows to increase the possible level of memory and CPU overcommitment
- **Linux suspend mechanism**
  - takes 10 - 30 sec to hibernate for a 850 MB guest, 20 to 30 sec to resume (1 – 5 GB guest)
  - controlled halt
  - the suspended guest is safe!
- **z/VM stop/begin command**
  - system reacts immediately
  - guest memory is lost in case of an IPL or power loss
  - still some activity for virtual devices
- **There are scenarios which are not eligible for guest deactivation (e.g. HA environments)**
- **For additional Information to methods to pause a z/VM guest**
  - http://www.ibm.com/developerworks/linux/linux390/perf/tuning_vm.html#ruis

Summary – scaling JVMs per guest

Question: One or many WAS JVMs per guest?

Answer:

- **In regard to throughput: it doesn't matter**
- **In regard to CPU load: it matters heavily**
  - Difference between maximum and minimum is about 4 IFLs(!) at a maximum total of 13.7 IFLs
  - The variation in CPU load is caused by User space CPU in the guest
  - z/VM overhead is small and has its maximum at both ends of the scaling
  - 2 virtual CPUs per guest provide the lowest CPU load for this workload
- **Sizing recommendation**
  - Do not use more virtual CPUs are required
  - If the CPU overcommitment level becomes not too high, use at minimum two virtual CPUs per WebSphere system
- **DCSS vs shared disk**
  - There is only a difference in CPU load, but that reaches the area of 1 - 2 IFLs
  - For less than 10 guests a shared disk is recommend
  - For more than 20 guests a DCSS is the better choice
- **For additional Information**
  - WebSphere JVM stacking: http://www.ibm.com/developerworks/linux/linux390/perf/tuning_vm.html#hv
  - page reorder: http://www.vm.ibm.com/perf/tips/reorder.html

Backup

Stop/Begin Test – What happens with the pages?