[April, 2009]

WebSphere Application Server V7 with z/VM 5.4.0

Performance and Scalability

Document version [1.01]

Jim Cunningham, IBM
William Scott, IBM
John Stecher, IBM

Contents

1

Executive Summary

z/VM and IBM WebSphere combine to create a high performance platform for server consolidation of enterprise applications on IBM System z.

The evolution of multicore technology and the ever increasing performance of modern day CPUs provides ever-expanding computing capabilities on a single hardware machine. This increase in computing power combined with a new focus on reducing energy and cooling costs is driving a mass migration towards server consolidation. Server consolidation in todays IT environments is focused on collapsing multiple underutilized physical servers onto a single high-performing hardware platform. Virtualization technology is commonly employed in combination with multicore platforms for server consolidation scenarios and provides a number of critical advantages. These include the ability to run multiple OSes and OS versions, application isolation, resource management, high availability, and more.

IBM z/VM and the IBM® WebSphere® Application Server software combine to create a high-performance platform for consolidation of enterprise application servers on ultra-robust mainframe hardware. z/VM 5.4.0 is the world's most powerful, scalable and reliable virtualization platform. When coupling the two products together, companies are able to build out cost-effective datacenters that allow their development and administration teams to meet the demands of their ever-changing business models quickly, efficiently, and most importantly, reliably.

This paper details performance and scalability results of WebSphere Application Server V7 running in z/Linux virtual images on top of z/VM 5.4.0. A number of server consolidation scenarios are described and compared with performance and scalability of native performance on a state-of-the art IBM System z10.

Introduction

Virtualization is the ability for a computer system to share resources so that one physical server can act as many virtual servers. z/VM allows the sharing of the mainframe's physical resources such as disk, memory, network adapters and CPUs. These resources are managed by a hypervisor. z/VM's hypervisor is called the Control Program (CP). When a user logs onto z/VM, the hypervisor creates a virtual machine which can run one of many different mainframe operating systems, like z/OS, z/TPF, Linux, z/VSE, CMS or even another z/VM.

Creating many virtual machines consisting of virtualized processors, communications, storage, and I/O devices can reduce administration costs and overhead of planning, purchasing, and installing new hardware to support new workloads. Through the "magic" of virtualization, software running within the virtual machine is unaware that the hardware layer has been virtualized. It believes it is running on its own hardware separate from any other operating system.

Why use z/VM to run Linux

Linux can run natively, right on the hardware or it can run under z/VM. The advantage of running natively is the elimination of the CPU overhead that it takes to run the VM, but there are disadvantages too. IBM System z hardware allows the creation of many logical partitions (LPARs) which are very suitable for running Linux. But you might have several lightly loaded Linux systems and you don't want to use all of your LPARs to do just this. Therefore, a good argument for running Linux under z/VM is that it conserves LPARs. A more important reason, however, is z/VMs ability to virtualize CPUs and memory and share those resources among many Linux guests. The result being that

you are able to run the same number of Linux guests on z/VM using fewer CPU and Memory resources than it would take to run them natively.

Running WebSphere on Linux under z/VM

This paper is meant to give the reader some understanding of the resource requirements of running WebSphere Application Server on Linux on z/VM. It details the performance and scalability result of focused research using WebSphere Application Server running on Linux as a guest of z/VM. A number of server consolidation scenarios are detailed, comparing virtual machine (VM) performance and scalability to a comparable, native set up. The native configuration employs the same OS and software stack, but is installed directly on the operating system without virtualization technology.

Testing methodology

Performance measurements were collected in both native and virtual environments. For the native scenarios, WebSphere Application Server V7 is run directly on the zLinux operating system (SLES 10.2). For the virtual tests z/VM is the base operating system, with SLES 10.2 and WebSphere Application Server V7 running in isolated virtual machines (VMs) with 2GB of virtual memory. Figure 1 illustrates this topology.

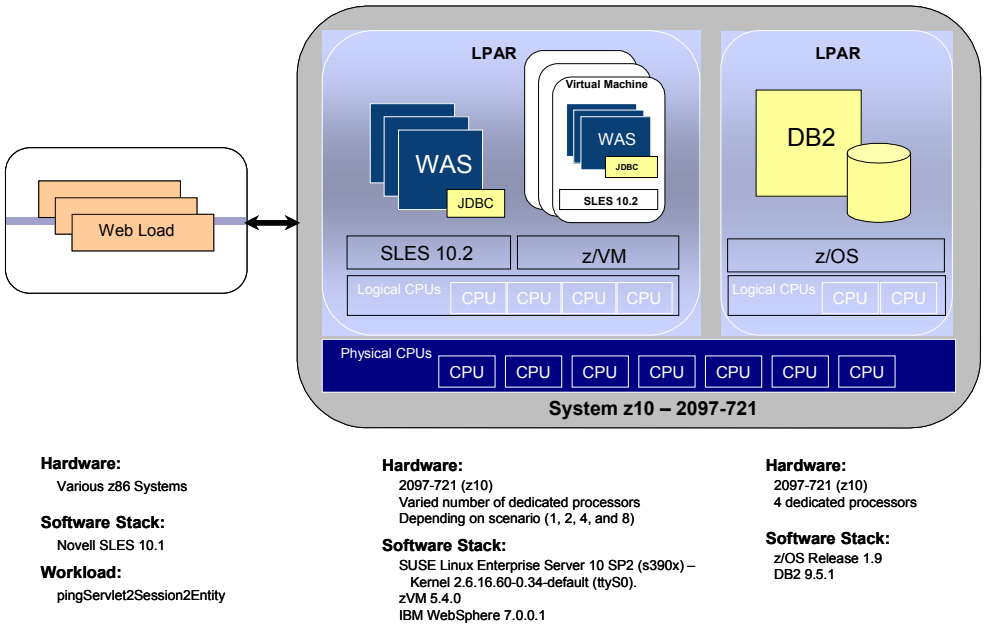

Figure 1 – Performance test topology

The WebSphere and database tiers reside on separate LPARs located on the same physical IBM System z10 server. Communications between the WebSphere tier and the database tier is done using HiperSockets. HiperSockets is a technology that provides high-speed Transmission Control Protocol/Internet Protocol (TCP/IP) connectivity between servers within an IBM System z mainframe. This technology eliminates the requirement for any physical cabling or external networking connections. It works similarly to an internal Local Area Network (LAN).

The Integrated Facility for Linux (IFL)

The Integrated Facility for Linux (IFL) is a central processor (CP) dedicated to Linux workloads. It is supported by z/VM, the Linux operating system and Linux applications. An IFL has the same performance and functionality of a full capacity, IBM System z central processor. Linux workload on an IFL does not result in increased IBM software charges for the Linux operating system and middleware. Also, many software vendors have adopted the IBM pricing model for Linux workloads. Since the performance of IFLs is identical to central processors and since the focus of this paper is purely on performance and scalability all measurements were conducted using CPs. Identical results would be achievable using IFLs.

The z/VM Memory Hierarchy

z/VM has a three level memory hierarchy: real storage (sometimes referred to as main storage, central storage or physical storage), expanded storage, and paging space (i.e. DASD or disk space). Most people are familiar with the definitions of real storage and paging space. Expanded storage is located in real storage but acts as a high speed paging device and is addressable in 4k pages, the same as DASD paging space.

Since expanded storage is carved out of real storage a logical question is: Why not just use all real storage and no expanded storage? The general recommendation is to configure z/VM with expanded storage. Here are a few reasons why.

- While configuring some expanded storage may result in more paging, it often results in more consistent or better response time. The paging algorithms in z/VM evolved around having a hierarchy of paging devices. Expanded storage is the high speed paging device and DASD the slower one used for block paging. This means expanded storage can act as a buffer for more active users as they switch slightly between working set sizes. These more active users do not compete with users coming from a completely paged out scenario.

- The real versus expanded issue is related to the different implementations

of LRU ("Least Recently Used") algorithms used between stealing from real storage and expanded storage. In real storage, z/VM basically just uses a reference bit which gets reset fairly often. While for expanded storage, z/VM uses an exact timestamp of a block's last use. This allows z/VM to do a better job of selecting pages to page out to DASD.

All measurements done for the memory over-commit scenarios were done using 8GB of real storage and 4GB of expanded. All other measurements were done using 16GB of real storage and 4GB of expanded.

DayTrader Benchmark Primitive

DayTrader 1.2 components
Web (servlet/JSP)
DataBase (EJB, JDBC)
Messaging (P2P, Pub/Sub)
Transactions (XA/2-phase across DB / Messaging)

The benchmark used to measure the performance and scalability is a fairly simple servlet/EJB primitive that is packaged with DayTrader 1.2. DayTrader is a Java Enterprise Edition (Java EE) application modeling an online stockbrokerage. Web users can login, view and modify their account information, check stock quotes, buy and sell shares, etc. In addition, it contains several Web and EJB primitives that are typically used to focus on the performance of specific Web/EJB functionality.

The primitive chosen for these tests is called *pingServlet2Session2Entity* and, as the name implies, it is an HTTP request that invokes a servlet that calls a session bean that calls an entity bean to read data from DB2. It is read-only and almost all the data is cached in WebSphere so the load on DB2 is very light. It was chosen because we mainly wanted to demonstrate the CPU and memory consumption of WebSphere running in a VM and did not care much about the interactions with DB2.

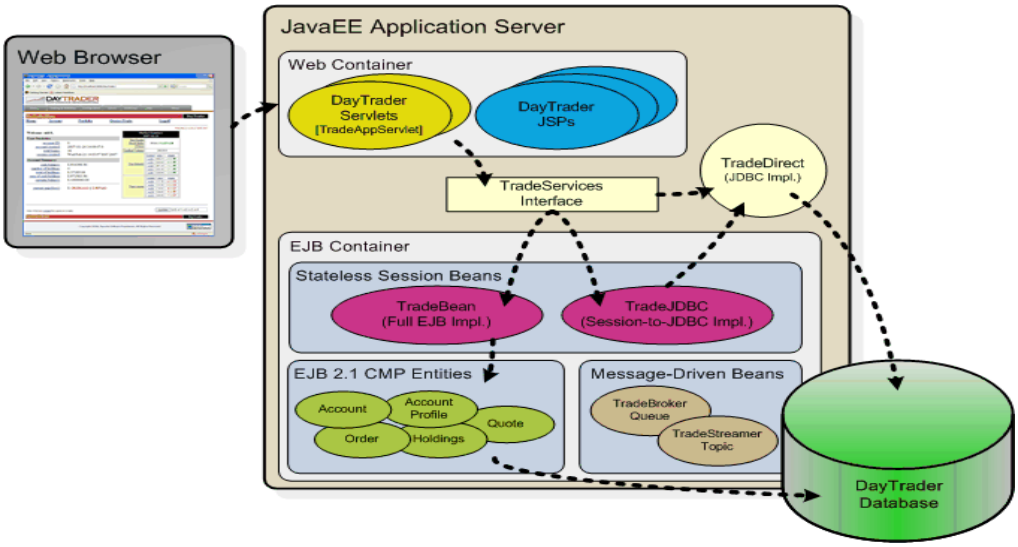

Figure 2 - DayTrader 1.2 architecture

IBM donated the benchmark to the open source community and it is now developed and maintained under the Apache Geronimo project. All source code and benchmark artifacts can be downloaded under the Apache open-source license. (http://geronimo.apache.org/)

Benchmark Comparisons

As is the case with every performance evaluation, the actual throughput or performance that any user will experience will vary depending upon many factors, including, but not limited to, the amount of multiprogramming, the I/O configuration, the storage configuration, and the workload mix. Therefore, no assurance can be given that an individual user benchmark will achieve results similar to those stated here.

3

Performance and Scalability

The goal for this research is to highlight the performance and scalability of WebSphere Application Server V7 in server consolidation scenarios using z/VM virtualization relative to native. A set of test scenarios has been created to measure WebSphere Application Server performance while varying the amount of physical resources available in equivalent native and virtualized environments. This section describes performance and scalablity with the server being driven to full utilization (CPU saturation). While full utilization is not typical in a server consolidation scenario, fully driving the server characterizes the limits of performance and scalability as well as the maximum overhead associated with virtualization.

Native and VM, Single Application Server

This section covers performance and scalability characteristics when running a single WebSphere Application Server process instance, both native and within a single VM. WebSphere Application Server is a multithreaded server that can scale to many CPUs with a single process instance. WebSphere Application Server supports server consolidation in this scenario through its ability to run and manage multiple applications within a single instance of the application server.

In the native configuration, a single WebSphere Application Server server was used. The LPAR was IPLed with one, two, and four logical CPUs. In the z/VM configuration, a single virtual machine was used, running one instance of WebSphere Application Server. The virtual machine was defined with one, two and four virtual CPUs enabled. There were an equivalent number of logical CPUs as virtual CPUs for each test. The throughput was measured and recorded at each point. In each of these instances, the server platform was driven as near

to 100% CPU saturation as possible.

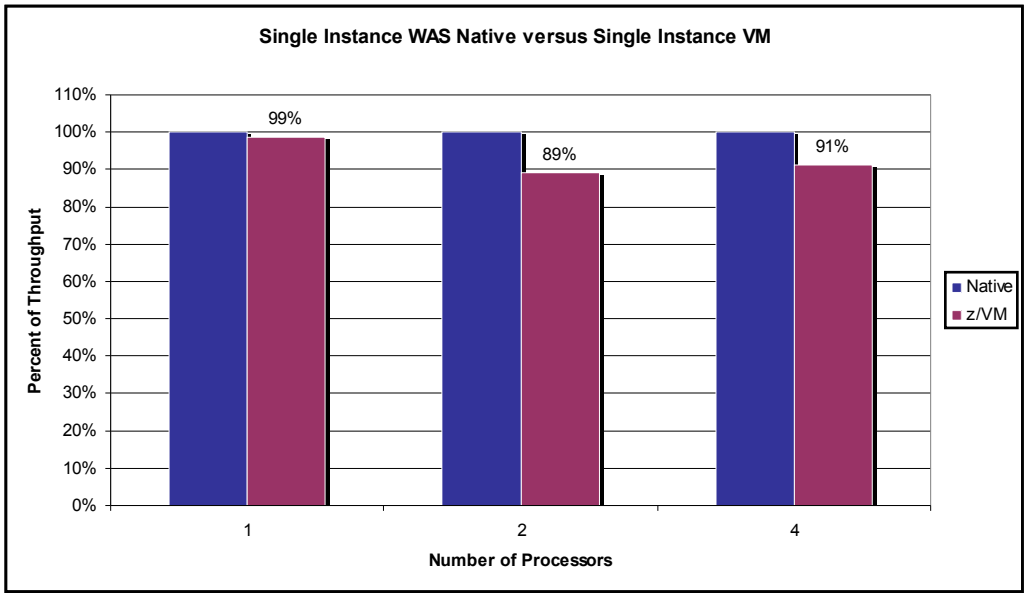

Figure 3 - Single VM relative to Single Instance native

Analysis

In Figure 3, the throughput for each data point in the native configuration is plotted as 100% and is compared relative to each point in the z/VM configuration. The results show that for a single processor the throughput achieved by the z/VM configuration was almost identical (-1%) to the native configuration. With 2 and 4 CPUs the delta increases to -11 and -9 percent respectively. The difference is due to the overhead introduced from the hypervisor, which manages virtual machines and their associated resources on the physical server.

Scalability with a Single Application Server instance

The previous figure depicts the performance gains for a single WebSphere Application Server instance as processors/vCPUs are enabled. Figure 4 plots these results as a scalability curve for both virtual and native scenarios. The scalability ratio is defined as the throughput achieved at *n* CPUs, divided by the throughput achieved with one CPU. This value effectively shows the percentage of total CPU time being used toward productive workload throughput, versus the amount consumed by system overhead as processors are added.

In multiprocessor systems, system performance is not expected to achieve an

exact linear improvement as the number of processors increase. This limitation is due to many factors, such as memory bus saturation, hardware cache utilization, software lock contention, and so forth. A key challenge for modern enterprise software is to efficiently scale to a large number of cores in evolving multiprocessor hardware designs.

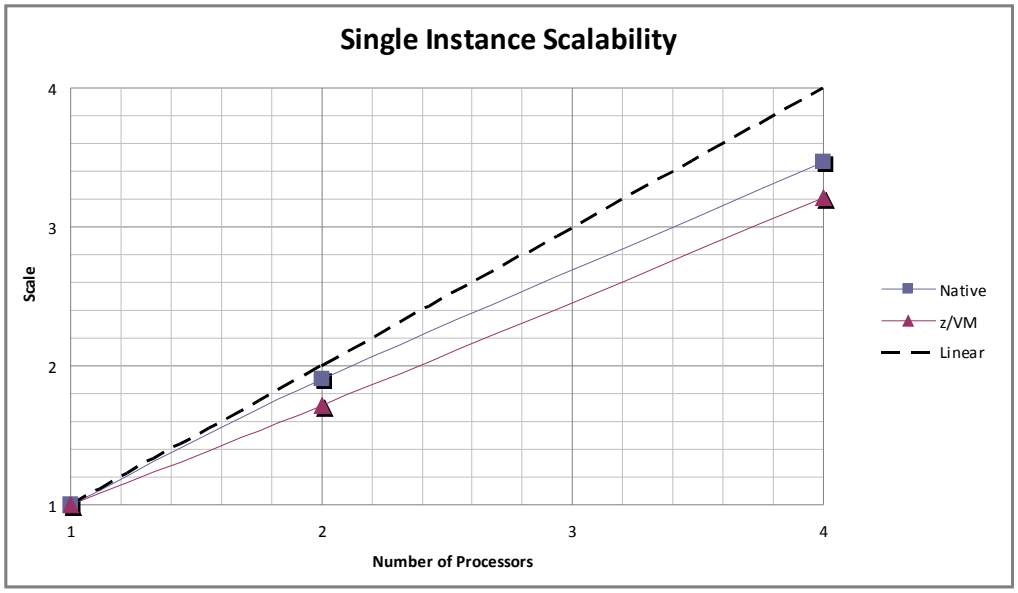

Figure 4 - WebSphere Application Server scalability

Analysis

The figure shows that native scalability is slightly better than z/VM as CPUs are ramped from one, to two and four. However both are very good. The native scalability ratio of 1 to 2 processors is about 1.9x, and for 2 to 4 processors about 1.8x. For z/VM these ratios are 1.7x and 1.9x respectively.

Consolidation and Over-provisioning

The previous section describes cases where a single application server is driven to full utilization of the underlying hardware. While this is good for understanding the general limits of scalability and overhead of virtualization, it is not the standard use case for server consolidation. The following sections detail performance using multiple instances and multiple virtual machines where each application is driven by a lightweight workload that only utilizes a fraction of the allocated resources. This is a simple model to emulate server consolidation where many under-utilized physical servers are collapsed into a single multicore server.

Benefits of consolidation

Consider for a moment the number of idle or under-utilized servers that might exist in a typical lab or data center. Each of these systems consumes power, rack space, and time in the form of maintenance and administration overhead. While it is costly to allow servers to remain idle, it's also unreasonable in most cases to power a system down. Consolidation through virtualization provides a solution by pooling hardware resources and scheduling them according to demand. If a VM has idle resources, they can be redirected to other systems where they are needed. Under this model the cost of idle servers can be minimized, while allowing their function to continue.

Lightweight workload conditions

Server consolidation is simulated in this experiment by allocating more virtual processors than are physically available on the hardware. While the concept of consolidation does not require system resources to be over provisioned, it does provide an opportunity to discuss the expected behavior of VMs in a dense environment. Instead of driving VMs to 100% CPU saturation, 40% workload is used to represent more typical traffic patterns. Figure 5 shows the total system

throughput as partially loaded 4-way/2GB VMs are incrementally added to the consolidated environment, hosted on a 8-way LPAR with 16GB of real storage and 4GB of expanded. The 4-way VM configuration was chosen because historically 4-way servers have been common in the data center, due to their desirable cost/performance ratio.

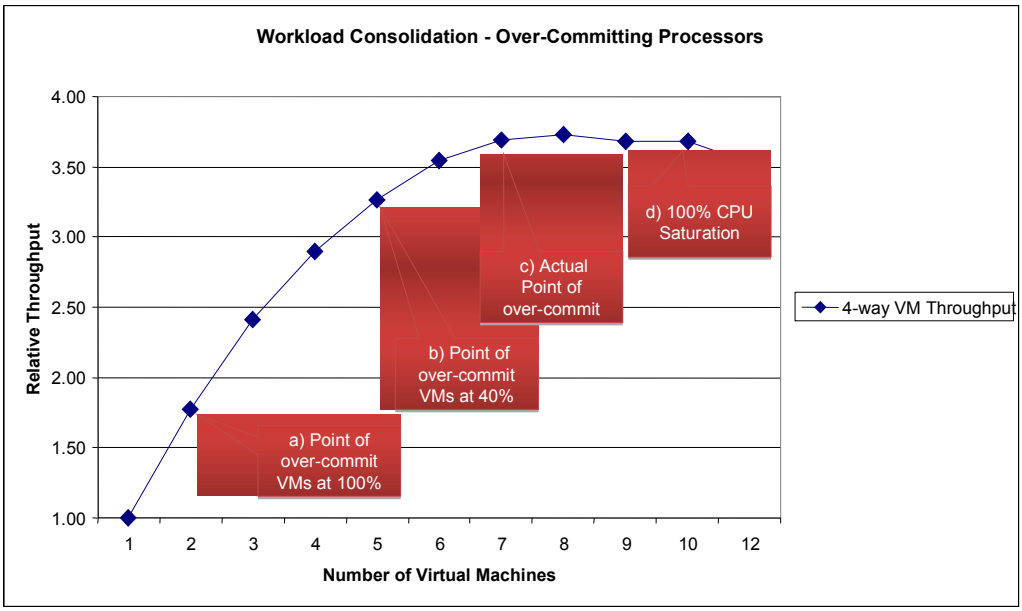

Figure 5 - 4-way virtual machine

Analysis

Most people automatically think the point of CPU over-commitment is the point at which the number of virtual processors outnumber the number of physical processors. But this is a little too simplistic since it assumes each VM is running at 100% CPU utilization. If this was the case then our over-commitment point would be after just 2 VMs (a). For our test, we drove the single VM case with a single client which resulted in about 40% CPU utilization. Subsequent VMs were also driven by a single client. Given there are 8 logical CPUs available to the hypervisor, we can expect the 4-way VMs under this load to achieve the theoretical maximum throughput when five VMs are running simultaneously (b). Each 4-way VM at 40% load is effectively consuming only 1.6 processors, leading to the system capacity of five VMs expressed in the following equation:

$$\frac{8\ Logical\ CPUs}{\frac{4\ vCPU}{VM} * 40\%} = 5.0\ VMs$$

However, this is the theoretical limit which assumes that, as more VMs are added, each VM will consume exactly 40% of available CPU. For our test, when there was only a single VM, it consumes about 40% of the CPU. However as more VMs were added, z/VM recognized that CPU resources were

becoming scarce and needed to limit the CPU consumption of each VM. Also, additional overhead was required by the hypervisor and the utilization of each VM decreased to an average of about 30%. At 30% load the expected saturation point is about 7 VMs which is what is observed in the figure 5 (c).

CPU Over commit

With every discussion of server consolidation, it's important to highlight the potential hazards of over provisioning physical resources such as processors and memory. While it saves energy and lab space to virtualize underutilized servers, there is always the potential for the wrong combination of virtual machines to reach peak capacity at the same time. If this occurs, the physical hardware supporting the virtual machines may not have enough resources to handle the workload. Figure 5 shows that z/VM is able to sustain roughly equivalent throughput for 8, 9 and 10 VMs. At 12 VMs, CPU resources have been completely saturated and throughput begins to decline (d).

CPU Time

Up until this point in the paper the focus has been on the throughput and scalability. Very little has been said about CPU consumption. Figure 6 shows the average amount of CPU time consumed per request as more VMs are added.

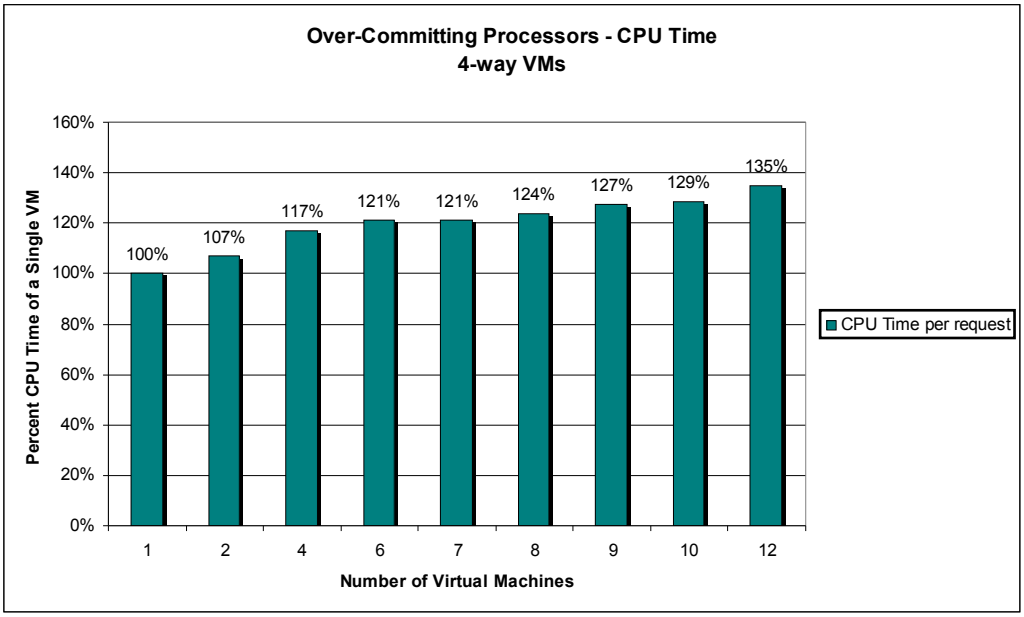

Figure 6 - CPU consumption - 4-way virtual machines

Analysis

Figure 6, shows the CPU consumption per request for each VM relative to the amount consumed by a single VM. As z/VM is required to service more VMs

contention for processors increases, more context switches occur and each VM's data is less likely to be found in the hardware cache which causes CPU time to increase. With 2 VMs the average amount of CPU time per request increases by about 7%. As more VMs are added this time increases, and for 12 VMs it has grown by about 35%.

Memory Over commit

The consequences of over provisioning memory are significantly more dramatic than for the CPU resource. Figure 7 details the observed performance while running multiple VMs on a server with only 12GB of memory (8GB real and 4 GB expanded). While this is a very small z/VM host, it effectively demonstrates performance expectations when memory capacity is limited.

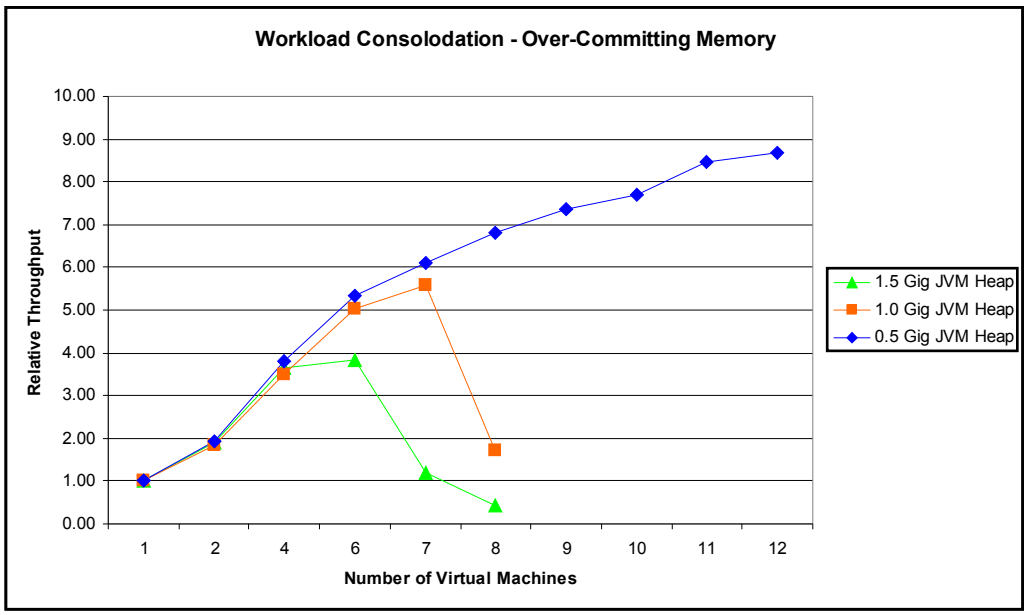

Figure 7 - Memory over commit on 12GB z/VM server

Three VM configurations, each with a different JVM heap size, have been plotted to demonstrate the impact of memory allocation decisions on system performance. For each configuration each VM is allocated 2GB of virtual memory.

For the the first series, each VM is assigned a 1.5GB Java heap. As the graph shows, there is sufficient real storage to contain up to, and including 4 VMs. After 4 VMs, z/VM starts paging to expanded storage and throughput begins to

level off. After 6 VMs expanded memory also becomes exhausted and z/VM starts paging to DASD. At this point performance drops dramatically. Similar to an OS and paging, when the hypervisor has exhausted all physical memory and VMs require system resources, data must be saved to disk in order to free memory for use by other VMs.

The second data series shows the memory over commit scenario where the JVM heap size is reduced to 1.0GB. Notice the shape of the graph is very similar to the shape of the graph for the 1.5GB heap. Paging to expanded storage begins at 6 VMs and throughput continues to improve for 7 VMs. At 8 VMs paging to DASD is required and, again performance drops significantly. With lighter memory demands from the VM and Java heap allocation, the hypervisor is able to service WebSphere Application Server requests for 2 more 1.0GB VMs before performance began to decline.

For the third data series the JVM heap size was reduced even further, down to 0.5GB. As the shape of the graph indicates, z/VM was able to contain up to 12 VMs in real storage. There was a very small rate of paging to expanded storage in this case but not nearly enough to effect performance.

Virtual machines and Java memory management

One of the key benefits of the Java platform is managed memory. Objects are allocated by applications in a heap structure which forms the main memory of a Java application. When the heap is close to full, the Java garbage collector (GC) automatically removes objects which are no longer in use by the program. This generally frees a large percentage of the heap memory space.

The runtime characteristics for the heap memory then are that it is continuously filled and then partially or mostly emptied with each GC. This is an important point relative to virtualization in that all of the heap memory is regularly "touched" by the application. The z/VM hypervisor generally attempts to share the server's physical memory among VMs, which can be done efficiently for memory that is rarely touched. However, Java heap memory cannot generally be shared between isolated VMs because it is dynamic and constantly being accessed.

In server consolidation scenarios, it may be beneficial in some cases to reduce the size of the Java heap for WebSphere Application Server applications. This will enable a larger number of applications to be consolidated in a memory constrained environment. There are tradeoffs however. Reducing the size of the heap will generally have a performance impact because GC cycles happen more frequently. If the heap size is reduced too much, runtime "out of memory" errors

can occur causing the application to halt. This will happen when the heap is not large enough to contain the application working set.

VM Working Set Sizes for WebSphere

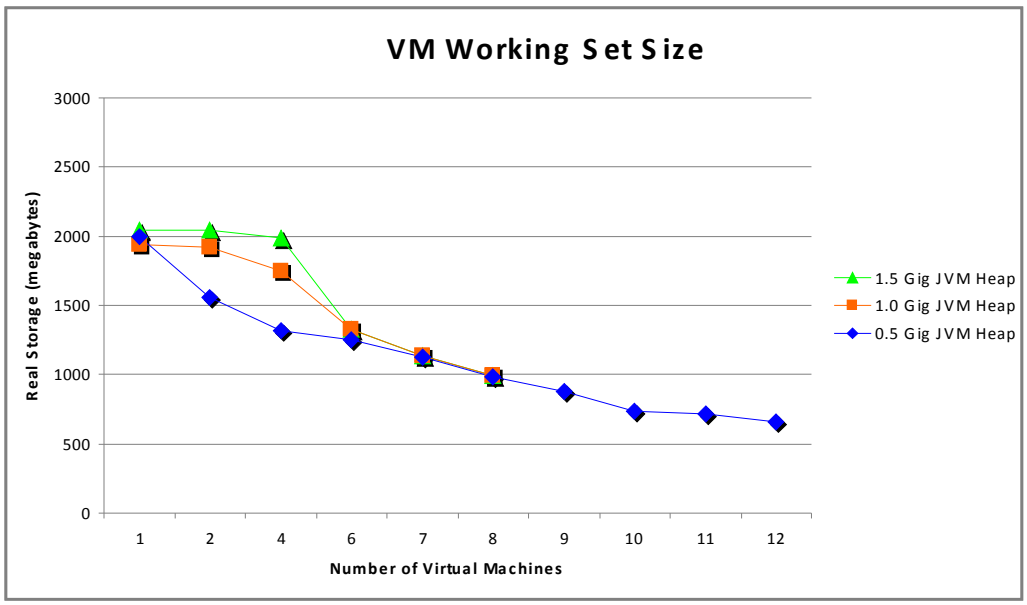

Figure 8 - VM Working Set Sizes

Analysis

When real storage is plentiful no paging is necessary and even infrequently referenced data remains resident in real storage. This fact is evident in Figure 8. When there is only a single VM the working set sizes for all three JVM heap sizes are about the same and almost equal to the 2GB virtual memory limit of the VM. The LPAR used for these tests was defined with 8GB of real storage, so at 2GB per VM, 4 VMs can reside in real storage. But the system requires some storage too so we start to see a slight decrease in working set size even for 2 and 4 VMs. At 6VMs the working set sizes for all three JVM heap sizes have converged.

Recall from the discussion about memory over-commitment and Figure 7 that, in the 1.5GB JVM heap case, throughput was good with 4 VMs and here we can see that the working set size was still almost 2GB. This is because all 4 VMs can fit into real storage and no paging is required. At 6 VMs, throughput has almost leveled off due to increased paging rates and we see that the working set size has dropped to about 1.3GB which is less than the JVM heap size. Then, with 7 VMs throughput has dropped dramatically and the working set size was reduced even further down to about 1.1 GB. From this we can conclude that, for this workload, storage configuration and JVM heap size, that about 6 VMs

can be supported while still achieving relatively good throughput. To reach higher throughput rates would require either reducing the JVM heap size or increasing the storage configuration.

We see a similar pattern for the 1.0GB JVM heap size. Throughput is good for 7 VMs where the working set size is 1.1GB which is still about 100MB greater than the JVM heap size. With 8 VMs throughput has declined significantly with a working set size of 1.0GB. So with a 1.0GB JVM heap size, this storage configuration can support up to 7 VMs before throughput rates decline.

For the 0.5GB JVM heap size, throughput continues to climb even with 12 VMs. At 12 VMs the working set size is about 650MB so all 12 VMs fit into real storage and no paging occurs.

So, again referring to figure 7, we can see that the maximum relative throughput achieved with a 1.5GB JVM heap was about 3.8 with 6 VMs. For a 1.0GB heap it was about 5.6 with 7 VMs and for a 0.5GB heap it was about 8.7 with 12 VMs. So by reducing the JVM heap size from 1.5GB to 0.5GB and not changing the storage configuration we were able to increase the relative throughput from 3.8 to 8.7 or by about 2.3x.

Performance Best Practices

This section provides some basic best practices for WebSphere Application Server V7 and z/VM performance.

- One of the leading causes of performance problems when running WebSphere on Linux on IBM System z is the incorrect allocation of memory in z/VM, Linux and WebSphere.
    - Only allocate what you need. Reduce your virtual machine memory requirement to the point where there is a small amount of page/swap activity rather than adding more real memory.
    - The JVM heap probably consumes the most amount of memory. Because Java constantly references the entire JVM heap it is best to make it as small as possible. You will need to monitor garbage collection cycles to determine if the JVM heap size is too small due to excessive GC activity.
    - A general rule of thumb is to add another 200MB to the optimal JVM heap size to estimate the total virtual machine memory requirement for a guest running WebSphere.
- The paging subsystem defined for this paper was not very sophisticated. However in the real world it can be critical as over-commitment levels increase.
- A good rule of thumb is to allocate twice as much page space as the aggregate virtual memory.
- Over provisioning CPU will generally result in an incremental performance loss when the total active load is close to the total available CPU resource.
- Don't allocate too many virtual CPUs. It is best to allocate the minimum number of virtual CPUs needed to get the job done.
- OS level performance statistics within a VM are not always accurate. Do

not rely on these statistics for tuning/management. Use the z/VM Performance Toolkit. It provides a wealth of performance data and the monitoring overhead is low.

S

Summary

Server consolidation on multi-processor hardware is growing significantly. IBM WebSphere and z/VM combine to create a high-performance platform for server consolidation of enterprise applications on IBM System z hardware. z/VM offers a base for customers who want to exploit IBM virtualization technology on one of the industry's best of breed server environments, the IBM System z family. With virtualization technology, customers can easily create many virtual machines consisting of virtualized processor, communications, storage, networking, and I/O resources.

Various scenarios were run showing performance and scalability of WebSphere Application Server V7 running with z/VM virtual machines compared to native. VM performance was generally within ~11% of native when running on a small number of processors.

Low load tests were also conducted to simulate server consolidation scenarios on multicore hardware. These tests show that several under-utilized application environments can be consolidated using WebSphere Application Server and z/VM technology into a single physical server. Over provisioning of resources with WebSphere Application Server V7 can have a significant effect on performance. Memory over-provisioning is the most critical for WebSphere Application Server V7, and too much can result in dramatic performance loss.

Other Sources of Information

1. For information on Linux on System z:
   http://www.ibm.com/systems/z/os/linux/
2. For information on WebSphere Application Server:
   http://www.ibm.com/software/webservers/appserv/was/
3. For information on WebSphere Application Server performance and tuning:
   http://www.ibm.com/software/webservers/appserv/was/performance.html
4. For information on IBM open-source projects:
   http://www.ibm.com/developerworks/opensource
5. For information on z/VM:
   http://www.vm.ibm.com

Acknowledgements

The following people are gratefully acknowledged for their contributions to this paper.

Romney White	IBM System z Virtualization Technology
Steve Wehr	IBM Systems &Technology Group, System z Platform
Steve McGarril	STG WW Mainframe Benchmark Center
Gene Ong	STG WW Mainframe Benchmark Center
Jay Parisi	Application and Integration Middleware Software
Julie Murphy	Application and Integration Middleware Software
Sue Lindsay	Application and Integration Middleware Software
Jim McNamara	Application and Integration Middleware Software
Joe DeLuca	IBM Systems &Technology Group, System Assurance
Ken Morris	Application and Integration Middleware Software
Aaron Quirk	Application and Integration Middleware Software
Stan Cox	Application and Integration Middleware Software