Linux on System z

Using the Linux cpuplugd Daemon to manage CPU and memory resources from z/VM Linux guests

Linux on System z

Using the Linux cpuplugd Daemon to manage CPU and memory resources from z/VM Linux guests

Note

Before using this information and the product it supports, read the information in “Notices” on page 63.

Contents

Figures

Tables

About this publication

This paper is intended to provide information regarding performance of environments using the cpuplugd daemon. It discusses findings based on configurations that were created and tested under laboratory conditions. These findings may not be realized in all customer environments, and implementation in such environments may require additional steps, configurations, and performance analysis. The information herein is provided "AS IS" with no warranties, express or implied. This information does not constitute a specification or form part of the warranty for any IBM® products.

Authors

Linux end-to-end Performance team:

- Dr. Juergen Doelle
- Paul V. Sutera

Acknowledgements

Thank you to the following people for their contributions to this project:

- Eugene Ong
- Stephen McGarril

The benchmarks were performed at the IBM System z® World Wide Benchmark Center in Poughkeepsie, NY.

Chapter 1. Introduction

An introduction to the cpuplugd daemon and what the tests described in this white paper set out to achieve.

Objectives

Sizing Linux z/VM® guests can be a complex task, despite the fact that z/VM does a good job of managing the resource requests as appropriately as possible. But oversized guests often cause additional management effort by the Hypervisor and undersized guests often have performance-related issues with workload peaks. A large amount of guests with large ratios of resource overcommitment (more virtual resources than are physically available) and changing workload characteristics over time make a correct sizing even more challenging.

Therefore to simplify guest management the obvious question is, why not let the system manage the resources automatically, based on the operating requirements of the guest. This ensures that each guest receives what it requires at a certain point in time and the limits can then be adjusted in cases where a guest unintentionally receives too many resources.

The Linux cpuplugd daemon, also called hotplug daemon, can control the amount of CPUs and memory available for a guest by adding or removing these resources according to predefined rules.

There is an updated version of the cpuplugd daemon available starting with SUSE Linux Enterprise Server (SLES) SP2 or Red Hat Enterprise Linux (RHEL) 6.2, which greatly enhances the capability to define rules and the available performance parameters for the rule set. This tool now provides exactly what is required to enable the operating system of the guest to manage the resources within the range of the guest definition.

This study analyzes various rules for the Linux cpuplugd daemon, which can be used to automatically adjust CPU and memory resources of a Linux z/VM guest. We used a development version from the s390 tools applied on a SLES11 SP1.

The methodology used is:

- Determine the performance of an manually optimized system setup and keep these sizings and performance as the baseline
- Then start with a common sizing for each guest of 4 CPUs and 5 GB memory
- Let the cpuplugd daemon to adjust the resources under a predefined workload
- Identify appropriate rules to minimize the resource usage with the lowest performance impact

This will help customers to automatically adjust and optimize the resource usage of Linux guests according to their current load characteristics.

Note: In the following paper are memory sizes based on 1024 bytes. To avoid confusion with values based on 1000 bytes, the notations are used according to IEC 60027-2 Amendment 2:

Table 1. Memory sizes

Symbol	Bytes
KiB	$1024^1 = 0$
MiB	$1024^2 = 1.048.576$
GiB	$1024^3 = 1.073.741.824$

That means one memory page has a size of 4KiB.

Executive summary

The approach used to analyze the impact of the cpuplugd rules.

This new version of the Linux cpuplugd daemon is a very powerful tool that can automatically adjust the CPU and memory resources of a Linux z/VM guest. Starting with a common sizing of 4 CPUs and 5 GB memory for each guest, it adjusts the resources as required. Guests with very different middleware and combinations of middleware, different memory sizes, and workload levels have been tested and compared with a manually sized setup.

The approach is to manage all the different guests with the same rule set. For CPU management the important criteria is the management target. It is possible to either manage the CPUs exactly and with a very fast response to changing requirements, or to have a system which reacts to increasing requirements in a very restrictive manner.

The more complex part with respect to one common rule set for all guests is the memory management. Here the requirements of the various guests were so different, that for one common set of rules a trade-off between best performance or minimal resource usage had to be made. Setting up individual rules could improve the result. However, even with the common set of rules the impact on performance can be kept small (around 4% throughput degradation and corresponding reduction in CPU load) with only 5% more memory (z/VM view) as compared to the manually-sized run.

In our memory management tests, we stopped the middleware to ensure that the memory was freed up and made available again. An alternative would be to set up the middleware so that the required memory buffers can shrink when not in use. For example, define a WebSphere® Application Server with a much smaller initial heap size than the maximum heap size.

Note: For a high performance environment it is recommended that the initial heap size is set to the maximum heap size to avoid memory fragmentation and memory allocations during runtime.

Another important aspect of the comparison is that the manual sizing requires a set of test runs to identify this setup, and it is only valid for that single load pattern. If, for example, the workload on one system increases and decreases on another system by a similar amount, the total performance will suffer, whereas the cpuplugd managed guests would activate and deactivate resources as required and keep the total amount of used resources constant, that is, without changing the resource overcommitment ratio. The system now reacts according to the load pressure.

Table 2 on page 3 compares the different approaches for sizing and their trade-offs:

Table 2. Different Sizing approaches and their trade-offs

Approach	**Effort**	**Sizing and performance**	**Flexibility**
Manual sizing	very high	optimal	none
Generic default rules	small	good trade-off	very high
Server-type adapted rules	singular effort	very good	high

The paper helps to select either generic rules or gives guidance to develop server type depending rules. The suggested default configuration file is described in "Recommended default configuration" on page 53.

This will help customers to automatically adjust and optimize resource usage of Linux guests according to the current load characteristics

Chapter 2. Summary

This summary discusses the elements of the cpuplugd configuration file in detail and the recommended settings.

See Appendix B, "cpuplugd configuration files," on page 53 for details about the tested configuration files. A sample config file named cpuplugd is available in `/etc/sysconfig` for each installation.

When testing the impact of various rules, it is recommended to avoid rules which cause a system to oscillate with a high frequency. This means that in a very short time period resources are repeatedly added and withdrawn. When the workload itself is oscillating with a high frequency, it might help to use average values over larger time periods and decrease the limits.

The opposite behavior is a very sensitive system which reacts very quickly to load changes to cover peak workloads. But even in this scenario, when these load peaks occur very quickly, it might be better to hold the resources.

Note:

- Managing the memory of a Linux guest with the cpuplugd daemon, rules out the usage of VM Resource Manager (VMRM) Cooperative Memory Management for this guest.
- Managing the CPUs of a Linux guest with the cpuplugd daemon is incompatible with task bindings using the **task_set** command or the cgroups mechanism.

CPU plugging

To vary the amount of active CPUs in a system, the CPUs are enabled or disabled via sysfs from the cpuplugd daemon.

Note: This changes the amount of CPUs within the range of CPUs defined to the guest, either via CPU statements in the user directory or via **CP DEFINE CPU** command.

-
```
UPDATE="1"
```

  The update parameter determines the frequency of the evaluation of the rules in seconds, 1 is the smallest value. We could not identify any overhead related to a 1 second interval. A larger interval would produce a system that reacts more slowly. The recommendation is to use 1 second intervals for a fast system reaction time. If the objective is not to react immediately to each value change from a certain parameter, the evaluation of that parameter might cover values from several intervals.

-
```
CPU_MIN="1"
CPU_MAX="0"
```

  These parameters define the range within cpuplugd daemon varies the amount of CPUs. The lower limit is set to 1, the maximum value is set to '0' which

means unlimited, so it is possible to use all of the CPUs that the guest is defined with. In case a middleware works better with two than with one CPU, CPU_MIN would be set to '2'.

- ```
  user_0="(cpustat.user[0] - cpustat.user[1])"
  nice_0="(cpustat.nice[0] - cpustat.nice[1])"
  system_0="(cpustat.system[0] - cpustat.system[1])"
  user_2="(cpustat.user[2] - cpustat.user[3])"
  nice_2="(cpustat.nice[2] - cpustat.nice[3])"
  system_2="(cpustat.system[2] - cpustat.system[3])"
  ```

  These rules calculate user, system, and nice CPU values from the last interval and the third previous interval. The `cpustat.<parm>` values are counting the CPU ticks for a certain type of load accumulated from the system start, that means they are continually increasing. At each update interval the parameters are determined and saved. They are referred to by an index, which starts at 0 for the most current value. The user CPU from the last interval is the difference between the most current user value `cpustat.user[0]` minus the value before `cpustat.user[1]`.

  Note: These values are accumulated values from all CPUs and counted in the number of CPU ticks spent for that type of CPU usage!

- ```
  CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
  CP_Active2="(user_2 + nice_2 + system_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
  ```

  The differences in CPU ticks for a certain type of load must be normalized with the total amount of CPU ticks in which they are gathered, because the length of the intervals always varies slightly.

  Note:

  – Even the `UPDATE` interval is specified with a fixed value in seconds, depending on the load level the real interval length might differ more or less. Therefore it is highly recommended to use the values from the `cpustat.total_ticks` array. The index has the same semantics as the values array, [0] is the most current value, [1] the one before, and so on.

  – `cpustat.total_ticks` values are accumulated CPU ticks from all CPUs since system start! If the system is 100% busy this means that the number of CPU ticks spent for user, system and nice is equal to `cpustat.total_ticks`.

  The actively used CPU value for this calculation is composed of the user, system, and nice CPU values.

- ```
  CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"
  ```

  We use the average of the current and the third previous interval to cover a certain near term interval.

-
  ```

```
idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"
CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"
```

The considerations are the same as for the elements which contribute to CP_ActiveAVG as described in the previous paragraph. The states idle and iowait contribute to idle.

Note: We did not include steal time in these formulas. Never count steal time as active CPU, because adding CPUs triggered by steal time will worsen the situation. In case of very high CPU overcommitment rates, it might make sense to include steal to idle and to remove a CPU if steal time becomes too high. This reduces the level of CPU overcommitment and allow for a low prioritized system to relieve the CPU pressure. For productive systems, we recommend that you ignore it, especially if it appears for a limited period only.

-
```
HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"
```

  – onumcpus is the current number of CPUs which are online.
  – Table 3 shows the interpretation of the variables CP_Active AVG and CP_idleAVG:

Table 3. Interpretation of the variables CP_Active AVG and CP_idleAVG

Variable	Range	Interpretation	Comment
CP_Active AVG	0-1	0 means: no CPU is doing work 1 means: all CPUs are actively used	includes user, system and nice
CP_idleAVG	0-1	0 means: no idle time 1 means: all CPUs are fully idling	includes idle and iowait, steal time is not included

(1-CP_ActiveAVG) represents the unused capacity of the system as value between 0 and 1. The multiplication with onumcpus creates a value with a unit in multiples of CPUs. Due to that the comparison with 0.08 refers to 8% of a single CPU independent to the size of the system.

The rules above:

– Add another CPU when only less than 8% of one (a single) CPU's capacity is available
– Remove a CPU when more than 1.15 CPUs are in the state idle or iowait.

This is the recommended CPU plugging setup for a fast reacting system. If a system that acts in a restrictive manner is required, a loadavg base rule as described in “DB2 UDB tuning” on page 51 can be used.

Memory plugging

To vary the amount of memory in a system, the cpuplugd daemon uses a ballooning technology provided by the Linux cmm module.

This manages a memory pool, called the CMM pool. Memory in that pool is 'in use' from the Linux operating system point of view, and therefore not available, but eligible for the z/VM in case memory pages are needed. The CMM module

handles the communication with the z/VM Hypervisor that these pages are disposable. To vary the amount of memory in a system, the memory is either assigned to or withdrawn from the CMM pool by the cpuplugd daemon.

-
```
pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"
```

  There are two mechanisms managing the memory, an asynchronous process, called the `kswap` daemon and a synchronous mechanism, called direct page scans. The kswap daemon is triggered when the amount of free pages falls below some high water marks. The synchronous mechanism is triggered by a memory request which could not be served. The last one delays the requester. We got very good results when using only the direct scans as in the following calculations. If this causes systems that are too small, `kswap` scans as used in configuration 3 in "Memory plugging configuration 3" on page 56 can be included.

-
```
pgscanrate="(pgscan_d - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
```

  Only the current situation is considered. If only direct scans are used as criteria this is important because the occurrence of direct page scans indicates that an application delay already occurred.

-
```
avail_cache="meminfo.Cached -meminfo.Shmem"
```

  The memory reported as cache consists mostly of page cache and shared memory. The shared memory is memory used from applications and should not be touched, whereas the page cache can roughly be considered as free memory. This is especially the case if there are no application runnings which perform a high volume of disk I/O transfers through the page cache.

-
```
CMM_MIN="0"
CMM_MAX="1245184"
```

  `CMM_MIN` specifies the minimum size of the cmm pool in pages. A value of zero pages allows the full removal of the pool. As maximum value (`CMM_MAX`) a very large value of 1,245,184 pages (4,864 MiB) was used, which would stop the size of the pool from increasing when less than 256 GB memory remain. In real life the pool never reached that limit, because the indicators for memory shortage were reached earlier and stopped the size of the pool from increasing.

-
```
CMM_INC="meminfo.MemFree  / 40"
CMM_DEC="meminfo.MemTotal / 40"
```

  These values are specified in pages (4 KiB each), KiB base values as the data in `meminfo` must be divided by a factor of 4, for example, 40 KiB is 10 pages. `CMM_INC` is defined as percentage of free memory (for example, 10%). This causes the increment of the CMM pool to become smaller and smaller the closer the

system comes to the 'ideal' configuration. CMM_DEC is defined for a percentage of the system size, for example, 10%. This leads to a relatively fast decrement of the CMM pool (that is, providing free memory to the system), whenever an indicator of a memory shortage is detected.

-
```
MEMPLUG   = "pgscanrate > 20"
MEMUNPLUG = "(meminfo.MemFree + avail_cache) > ( meminfo.MemTotal / 10 )"
```

  Memory is moved from the CMM pool to the system (plugged), when the direct scan rates exceed a small value. Memory is moved from the system to the CMM pool (unplugged), if more than 10% of the total memory is considered as unused, this includes the page cache.

  Note:

  – The increments for the CMM pool are always smaller then the smallest value created here to allow an iterative approach to reduce the volatile memory.
  – In case a workload depends on Page Cache caching, such as a database performing normal file system I/O, an increase of the limit specified in the MEMUNPLUG rule could improve the performance significantly. For most application caching behavior add twice the I/O throughput rate (read + write) in KiB as start value to the recommended 10% in our rule. For example, for a total throughput of 200MB/sec:

    ```
    MEMUNPLUG="(meminfo.MemFree+avail_cache)>(400*1024+meminfo.MemTotal/10)"
    ```

    In case of special page cache demanding applications even higher values might be required.

  For small systems (<0.5 GB) the 10% limit might be reduced to 5%.

Memory hotplug seems to be workload dependent. This paper does give a basis to start from, a server type-dependent approach for our scenario could look as follows:

Table 4. Server type-dependent approach

Server type	Memory size	CMM_INC	Unplug when
Web Server	< 0.5 GB	free mem /40	(free mem + page cache) > 5%
Application Server	< 2 GB	free mem /40	(free mem + page cache) > 5%
Database Server	= 0.5 GB	(free mem+page cache) /40	(free mem + page cache) > 5%
Combo	> 2GB	free mem /40	(free mem + page cache) > 10%

Installation of z/VM APAR VM65060 is a requirement when memory management via cpuplugd is planned. It reduces the amount of steal time significantly, more details are in “Memory plugging and steal time” on page 47. It is available for z/VM 5.4, z/VM 6.1, and z/VM 6.2.

Chapter 3. Hardware and software configuration

To perform our tests, we created a customer-like environment. This section provides details about the hardware and software used in our testing.

Server configuration

Server hardware

System z

One z/VM LPAR on a 56 way IBM System z196 EC, 5.2 GHz, model 2817-M80, equipped with:

- Up to 8 physical CPUs dedicated to z/VM
- Up to 20 GB Central memory and 2 GB Expanded Storage
- Up to 2 OSA cards (1 shared for admin LAN, and a 1-Gigabit OSA card.)

Storage server setup

The storage server was a DS8300 2107-932. For all System z systems and applications on up to 16 Linux host systems:

- 214 ECKD™ mod 9s spread over 2 Logical Control Units (LCUs)

Server software

Table 5. Server software used for cpuplugd daemon tests

Product	Version and release
IBM DB2 Universal Database™ Enterprise Server	9.7 fixpack 4
SUSE Linux Enterprise Server	SLES 11, SP1 64-bit + development version of s390-tools package
DayTrader Performance Benchmark	Version 2.0 – 20080222 build
WebSphere Application Server	7.0 fixpack 17, 64-bit
IBM HTTP Server	7.0 fixpack 17, 64-bit
z/VM	6.1

A development version was installed for the updated cpuplugd server.

Client configuration

Client hardware

Two IBM xSeries X336 2-way 3.60GHz Intel 8GB RAM were used as a DayTrader workload generator.

Client Software

Table 6. Client software used for cpuplugd daemon tests

Product	Version and release
WebSphere Studio Workload Simulator	Version 03309L
SUSE Linux Enterprise Server	10 SP2 (x86_64)

Chapter 4. Workload description

This section describes the following products that were used in the tests:

- "DayTrader"
- "WebSphere Studio Workload Simulator" on page 14

DayTrader

An internally available version of the DayTrader multi-tier performance benchmark was used for the cpuplugd studies.

DayTrader Performance Benchmark is a suite of workloads that allows performance analysis of J2EE 1.4 Application Servers. With Java classes, servlets and ServerPage (JSP) files, and Enterprise JavaBeans (EJBs) all of the major J2EE 1.4 application programming interfaces are exercised so that scalability and performance can be measured. These components include the Web container (servlets and JSPs), the EJB container, EJB 2.0 Container Managed Persistence, JMS and Message Driven Beans, transaction management and database connectivity.

The DayTrader structure is shown in Figure 1.

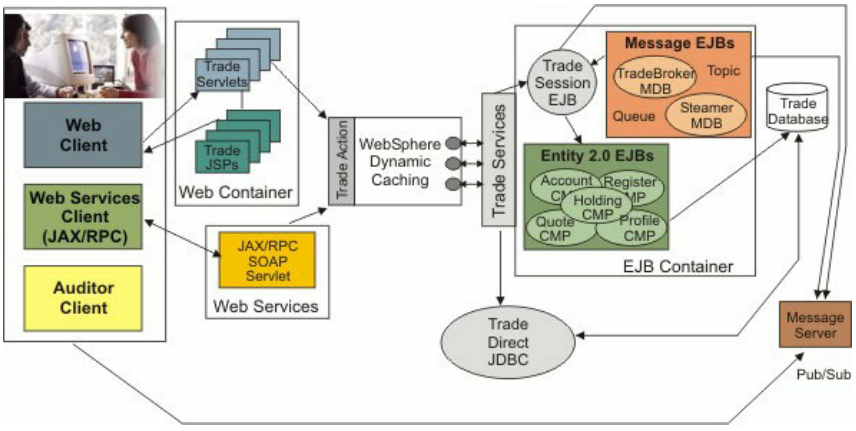

Figure 1. DayTrader J2EE components

DayTrader is modeled on an online stock brokerage. The workload provides a set of user services such as login and logout, stock quotes, buy, sell, account details, and so on, through standards-based HTTP and Web services protocols such as SOAP and WSDL. DayTrader provides the following server implementations of the emulated "Trade" brokerage services:

- EJB - Database access uses EJB 2.1 methods to drive stock trading operations
- Direct - This mode uses database and messaging access through direct JDBC and JMS code

Our configuration uses EJB 2.1 database access including session, entity and message beans and not direct access.

DayTrader also provides an Order Processing Mode that determines the mode for completing stock purchase and sell operations. Synchronous mode completes the order immediately. Asynchronous_2-Phase performs a 2-phase commit over the EJB Entity/DB and MDB/JMS transactions.

Our tests use synchronous mode only.

DayTrader can be configured to use different access modes. This study uses standard access mode, where servlets access the enterprise beans through the standard Remote Method Invocation (RMI) protocol.

Type 4 JDBC connectors are used with EJB containers to connect to a remote database.

To learn more about the DayTrader performance benchmark, or to download the latest package, find the DayTrader sample application at:
`HTTP://cwiki.apache.org/GMOxDOC22/sample-applications.html`

WebSphere Studio Workload Simulator

The DayTrader workload was driven by the WebSphere Studio Workload Simulator and the WebSphere Studio Workload Simulator script provided with DayTrader.

You specify the parameters for this script in a configuration file. Typically, you set up several different configuration files and then tell the script which file to use. The configuration changes we made are detailed in WebSphere Studio Workload Simulator configuration.

We used different copies of the modified WebSphere Studio Workload Simulator script to perform runs that were intended to stress anywhere from one to five application servers.

When DayTrader is running, four different workloads are running: two triplets and two combination mode servers. The stress test involves two clients for workload generation. Each client runs one DayTrader shell script, and each script invokes two separate instances of the iwl engine. One client targets the two combination servers, and the other client targets the two triplets, to spread the workload fairly evenly across the client workload generators.

Chapter 5. z/VM and Linux setup

This topic details the modifications we made to the system setup for our z/VM and Linux environments.

WebSphere environment

To emulate a customer-like configuration, one WebSphere Application Server environment consisted of:

- An IBM HTTP web server
- The WebSphere Application Server
- A DB2® UDB database server

This environment is called a "triplet".

"Combination" servers were also employed where the IBM HTTP Server, WebSphere, and DB2 UDB coexist on the same Linux on System z server, these servers are called "combos". The setup is shown in Figure 2. This setup represents guests with very different resource requirements at very different workload levels.

For the static portion of our test we use two triplets, and two combo servers; Triplet 1, Triplet2, Combo1 and Combo2. This is referred to in the paper as the first "Set" or "Set 1".

For the dynamic portion of the test, two additional triplets and two additional combo servers were added; Triplet 3, Triplet 4, Combo3 and Combo4. In this paper this is referred to as the second set or "Set 2".

For the static tests there are a total of eight Linux on System z guests. For the dynamic tests the total number of Linux guests is 16.

For dynamic workload testing the workload was switched to the second set after completing the workload on the first set, and then similarly back to the first set after completing the workload on the second set. This creates warmed up idling systems, which are switched from a state where the resource utilization is at its maximum to a state where the resource utilization is at its minimum.

For all tests we used two Client drivers on two System x336 systems.

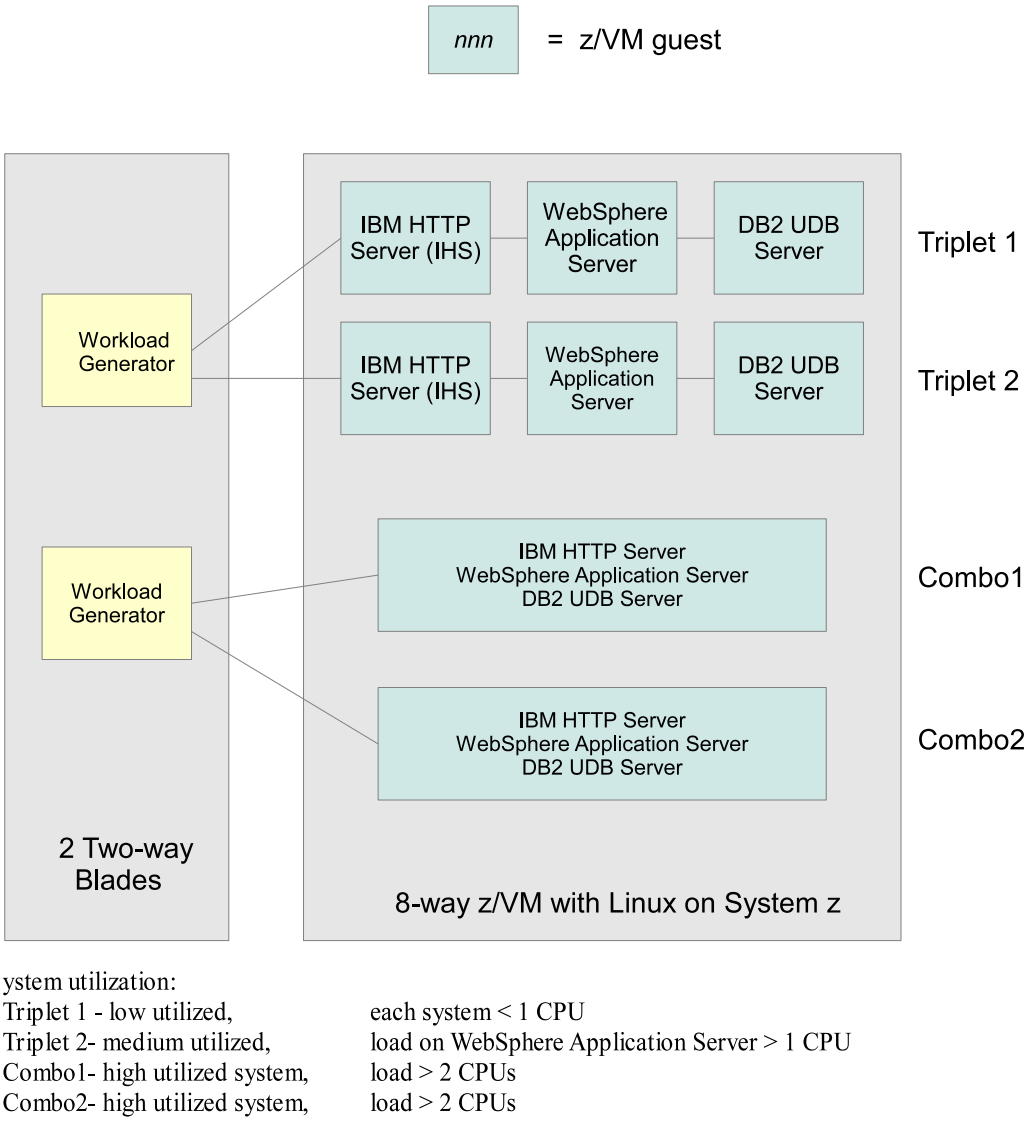

Figure 2. Setup for one set of servers and a targeted load

WebSphere Studio Workload Simulator configuration

How the default WebSphere Studio Workload Simulator configuration was modified for use in the tests is described in this topic.

Parameter changes

The parameter configuration file we passed to the workload generator engine was a modified version of the default WebSphere Studio Workload Simulator configuration provided with the DayTrader distribution. We specified the following parameter values:

- The number of simulated clients was set to four on the combo servers, three on Triplet 1 and 6 on Triplet 2. This number of clients enabled us to reach optimal throughput while keeping the system CPU between 90% and 97%.
- The time limit (length of our runs) was set to 10 minutes.
- The "element delay" (or "think time") was kept at 0.
- The "xml_interval" is the interval, in minutes, between successive snapshots of the output of the WebSphere Studio Workload Simulator. This output can be customized. Taking into account pages per second, transactions per second, and response time, we set the **xml_interval** to 5 minutes.

Below is a sample invocation of a WebSphere Studio Workload Simulator script for one triplet:

```
/var/iwl/bin/iwlengine -c 4 -e 0 -D on -r 10000 --enginename triplet1 --max_clients \
300 --xml_interval 5 --timelimit 600 -s /etc/iwl/common/trade6_lnweb1.jxs
```

Java heap size

A Java heap size of 1 Gigabyte was used for all WebSphere JVM Heap sizes and all test runs, with both the minimum and maximum heap sizes being set to 1 Gigabyte (1024 MB or 1024M).

Database configuration

The database buffer pools and other parameters are defined as shown in the tuning script in "DB2 UDB tuning" on page 51.

z/VM settings

This section describes the **Quickdsp** and SRM settings:

- "Quickdsp"
- "SRM settings"

Quickdsp

The set **Quickdsp** command and the `quickdsp` operand of the option directory statement allow you to designate virtual machines that will not wait in the eligible list when they have work to do.

All measurements in this study that were run on z/VM used the `quickdsp` operand. For the guests it was specified in the option directory statement for all z/VM virtual guest user directory definitions.

SRM settings

For some of the tests, the z/VM SRM settings were changed. The presence of the `quickdsp` option directory statement, however, may make these changes less relevant to the Linux virtual systems. Other z/VM virtual users dispatching priorities would be affected by the SRM settings because they did not run with the `quickdsp` option in our test.

It is recommended to use **CP SET SRM STORBUF** to increase the z/VM system's tolerance for over-committing dynamic storage. **CP SET SRM LDUBUF** is often used to increase the z/VM system's tolerance for guests that induce paging.

Some of the tests used the SRM values shown in the following **QUERY SRM** command:

```
q srm
IABIAS : INTENSITY=90%; DURATION=2
LDUBUF : Q1=300% Q2=300% Q3=300%
STORBUF: Q1=300% Q2=300% Q3=300%
DSPBUF : Q1=32767 Q2=32767 Q3=32767
DISPATCHING MINOR TIMESLICE = 5 MS
MAXWSS : LIMIT=9999%
...... : PAGES=999999
XSTORE : 0%
LIMITHARD METHOD: DEADLINE
Ready; T=0.01/0.01 15:35:12
```

The LDUBUF parameters specify the percentage of paging exposures the scheduler is to view when considering adding or loading a user into the dispatch list with a short, medium or long-running transaction, respectively. The values Q1, Q2, and Q3 shown in the output above, refer to the expected length of a transaction, where:

- Q3 is the longest running transaction
- Q2 includes medium and long length transaction users
- Q1 includes all users

The larger the percentage allocated, the more likely it is that a user is added to the dispatch list of users that are already on the list and waiting to be dispatched. Values over 100% indicate a tolerance for an overcommitment of paging DASD resources. A value of 300%, for example, indicates that all users in that transaction-length classification will be loaded into the dispatch list even if this would lead to an overuse of paging resources by up to three times.

The STORBUF parameters are also specified in three values. The values specify the percentage of pageable storage that can be overcommitted by the various classes of users (Q1, Q2, Q3) based on the length of their transactions, as described in the previous paragraph. Again, Q1 includes all classes, and Q2 includes Q2 and Q3 users. Q3 is reserved for long-running transaction users. Any value over 100% represents a tolerance for that amount of storage overcommitment to users included in that classification.

For some tests we changed the values using the **SET** command, as shown below, to set slightly less aggressive dispatching decisions for our non-Linux z/VM guests.

```
SET SRM LDUBUF 100 100 100
LDUBUF : Q1=100% Q2=100% Q3=100%
Ready; T=0.01/0.01 15:35:22

set srm storbuf 300 250 200
STORBUF: Q1=300% Q2=250% Q3=200%
Ready; T=0.01/0.01 15:36:19
```

Linux guests

This topic describes the baseline settings for Linux guests and which rpms are installed.

Baseline settings

The baseline CPU and memory settings for the Linux guests were established.

A DayTrader workload is tested against each triplet and each combo server through an iterative process to reach load targets and optimize sizing and resource usage.

A specific load based on a number of DayTrader clients running on each triplet and each combo server is established where system-wide CPU-utilization is between 90% and 97%. Baseline memory and CPU settings, in conjunction with the established DayTrader load, represent a hand-tuned end-to-end system where the virtual CPUs and memory allocated for each guest are just sufficient to support the workload.

The number of clients for each triplet and each combo server is then kept the same throughout the tests, keeping the workload generation constant. The baseline settings are a good approximation of the ideal CPU and memory configuration for a given client-driven workload.

In the case of the dynamic runs, a second set of servers was created as an exact replica of the first set of servers. Set 2 servers were only used for the dynamic test runs and are not up during the static tests when the configuration files are evaluated.

For the tests with cpuplugd, the guest definitions were changed to 4 CPUs and 5 GB memory for all guests. Using different cpuplugd configuration files and keeping the number of DayTrader clients constant we measured how the management of CPU or virtual memory or both for systems with oversized virtual resource allocations was affected by CPU plugging and memory plugging.

Various configuration files were tested for their ability to quickly and correctly adjust virtual CPUs and memory for a particular DayTrader user count, where correctly allocated memory is as close as possible to the manually-sized configuration.

Linux service levels

The following rpms were installed during the testing of the cpuplugd daemon on top of the SUSE Linux Enterprise Server (SLES11) SP1 distribution to fix known issues relevant for this test. The s390-tools rpm contained the new version of cpuplugd:

```
kernel-default-2.6.32.43-0.4.1.s390x.rpm
kernel-default-base-2.6.32.43-0.4.1.s390x.rpm
kernel-default-devel-2.6.32.43-0.4.1.s390x.rpm
kernel-default-man-2.6.32.43-0.4.1.s390x.rpm
kernel-source-2.6.32.43-0.4.1.s390x.rpm
s390-tools-1.8.0-44.45.2cpuplug8.s390x.rpm
```

Chapter 6. Results

This topic describes not only the test results but also the methods we used to set up and run the tests together with any observations and conclusions.

Methodology

To compare the impact of the various rules for the cpuplugd daemon, a manually optimized setup was prepared.

The objectives for the manual sizing were minimal memory requirements (without swapping) and a CPU utilization of 90% to 95% for all CPUs. The assumption was that the automated sizing using cpuplugd will never create a system with a higher utilization.

Compared to this workload, the additional CPU utilization needed to run cpuplugd, which is activated at intervals potentially as small as 1 second, was expected to be small, and the system will never run faster when the resources are managed by any kind of tool. The question of interest was whether the expected degradation would be large or acceptable.

Manual sizing

The workload was adjusted so that the individual servers have different requirements (100% = 1 CPU fully utilized)

- The web server systems are minimally utilized systems
- The database servers are also considered to be low-utilization systems, which means the load is always less than 90% of one processor
- The two standalone WebSphere Application Servers were used to vary the load:
  - Triplet 1, the WebSphere guest must not exceed 80% CPU utilization
  - Triplet 2 has a WebSphere server with 2 CPUs, and the workload is set so that the CPUs are utilized to approximately 130%
- The workload of the combo servers is set so that the CPUs are utilized to approximately 130% . In both cases the "last" CPU may be idle during the test

Table 7 shows the sizing settings as configured in our manual sized setup:

Table 7. Baseline Virtual CPU and virtual memory settings for set 1 and set 2

Set number	Guest name	Function	Number of CPUs	Memory (MiB)
1	lnweb1	IBM HTTP Server	1	342
1	lnweb2	IBM HTTP Server	1	342
1	lnwas1	WebSphere Application Server	1	1600
1	lnwas2	WebSphere Application Server	2	1600
1	lnudb1	DB2 UDB	1	512
1	lnudb2	DB2 UDB	1	512
1	lncombo1	All above	3	2300
1	lncombo2	All above	3	2300
2	lnwas3	WebSphere Application Server	1	1600

Table 7. Baseline Virtual CPU and virtual memory settings for set 1 and set 2 (continued)

Set number	Guest name	Function	Number of CPUs	Memory (MiB)
2	lnwas4	WebSphere Application Server	2	1600
2	lnudb3	DB2 UDB	1	512
2	lnudb4	DB2 UDB	1	512
2	lncombo3	All above	3	2300
2	lncombo4	All above	3	2300
2	lnweb3	IBM HTTP Server	1	342
2	lnweb4	IBM HTTP Server	1	342

The total memory size of all guests in a set is 9,508 MiB. Both sets together are defined with 19,016 MiB.

Monitoring the management behavior of the cpuplugd

Put your short description here; used for first paragraph and abstract.

To monitor the management decisions of the cpuplugd daemon, it is started with the option `-V` and `-f` to generate a log file, for example:

```
cpuplugd -c <config file> -f -V>&<logname> &
```

The messages in the log file are then parsed for the timestamp, the value for **`onumcpus`**, and the amount of pages in the statement “changing number of pages permanently reserved to nnnnn”. These numbers are used to determine the real system size.

Understanding the sizing charts

When the system resources are managed by cpuplugd the amount of CPUs and memory varies over time. The tables list the average memory sizes allocated at the time when the system's CPU load reaches a steady state.

To show the dynamic behavior, the charts depict the individual values over times of interest. However, these charts are not always clear. Figure 3 on page 23, as an example, shows the number of CPUs over time assigned to all guests.

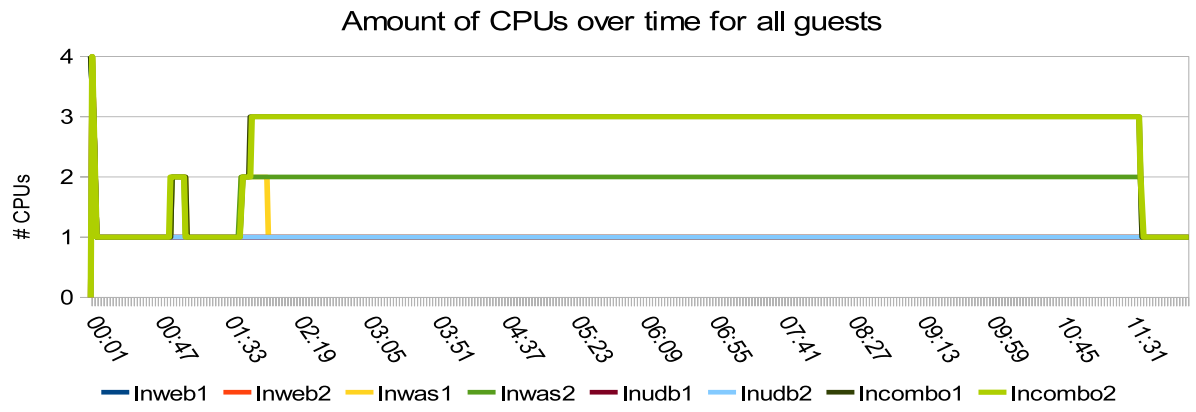

Figure 3. Number of active CPUs assigned to all guests when managed by cpuplugd

It is very hard to see what really happens. With the following simplifications:

- All systems which are expected never to exceed the load of 1 CPU are withdrawn. In the example, these are lnweb1, lnweb2, lnudb1 and lnudb2.
- The combo systems are expected to behave the same, therefore only lncombo1 is shown.

The result is shown in Figure 4.

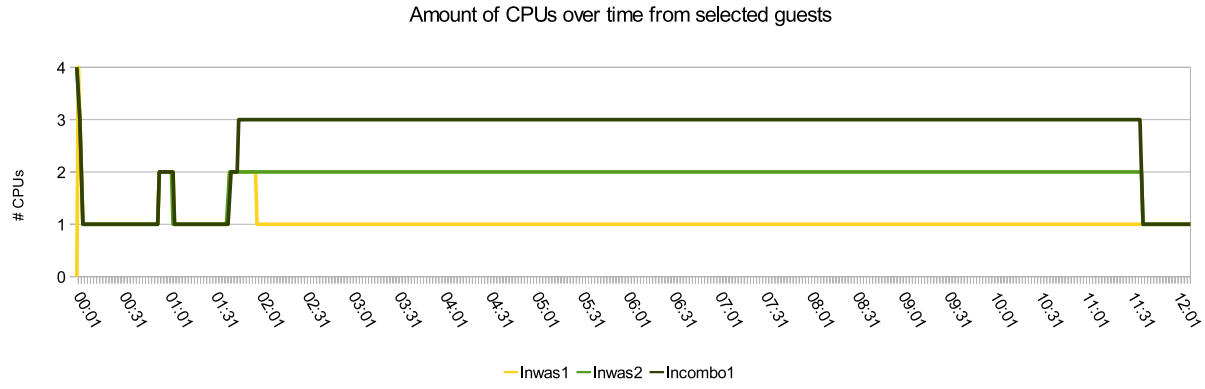

Figure 4. Number of active CPUs assigned to selected guests when managed by cpuplugd

After applying these simplifications you can see that the two WebSphere servers are handled differently. The chart also shows what happens during the different load phases:

00:00 cpuplugd gets started, the number of CPUs is quickly reduced to 1

00:29 The middleware is started, causing a short load increase to 2 CPUs.

01:41 The workload starts, after a ramp-up phase the number of CPUs assigned to lnwas1 is reduced

11:41 The workload stops, and the number of CPUs assigned to all servers is reduced to one

The charts for memory sizing are optimized in a similar fashion. The optimization rules are explained for each scenario. The memory size considered is the difference of the defined guest size, (5 GB per guest) and the size of the CMM pool, which is reported in `/proc/sys/vm/cmm_pages` (in pages). Mostly only the size of the CMM pool is shown; but a large pool signifies a small system memory size. The target is to be close to the manually sized setup. If the remaining memory size is smaller than the manually sized setup, it is likely that the performance is negatively impacted.

cpuplugd configuration rules

This topic describes the impact of various rules for CPU and memory management with cpuplugd.

CPU plugging

For managing CPUs with the cpuplugd daemon two rules are compared.

The two rules are:

- Using the first load average from `/proc/loadavg` (loadavg parameter). which is the number of jobs in the run queue or waiting for disk I/O (state D) averaged over 1 minute.
- Using averages of the real CPU load from the last three values

loadavg-based

The full configuration file is listed in Appendix B, "cpuplugd configuration files," on page 53.

The lines of interest are:

```
HOTPLUG="(loadavg > onumcpus + 0.75) & (idle < 10.0)"
HOTUNPLUG="(loadavg < onumcpus - 0.25) | (idle > 50)"
```

These lines implement the following rules:

- The system plugs CPUs when there are both more runnable processes and threads than active CPUs and the system is less than 10% idle
- The system removes CPUs when either the amount of runnable processes and threads is 25% below the number of active CPUs or the system is more than 50% idle

Figure 5 on page 25 shows the effect of the CPU management rules over time:

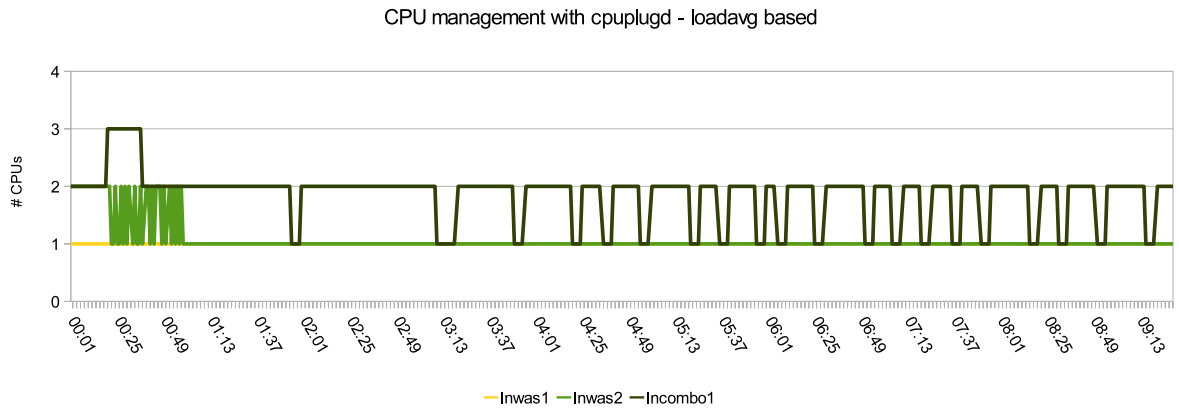

Figure 5. Number of active CPUs over time when managed by cpuplugd based on loadavg value

Table 8. Throughput and average CPU load when managed by cpuplugd based on loadavg value

Configuration	TPS*	Relative CPU load*
loadavg-based	88%	84%
*100% is the manual-sized run		

Observation

The number of CPUs is lower than manually sized for most of the time. The combo-system is frequently reduced to 1 CPU for a short time. The throughput is significantly reduced.

Conclusion

The value of `loadavg` determines the amount of runnable processes or threads averaged over a certain time period, for example, one 1 minute. It seems that this value changes very slowly and results in a system running short on CPUs. The guests running a WebSphere application server are always highly utilized, but the rules do not add the required CPUs, which leads to the observed reduction in throughput.

This configuration is probably useful when trying to restrict the addition of CPUs and to accept that the guests run CPU-constrained with the corresponding impact on throughput, for example in an environment with a very high level of CPU overcommitment.

Real CPU load-based

The full configuration file is listed in Appendix B, “cpuplugd configuration files,” on page 53.

This configuration uses the CPU load values (from `/proc/stat`). The values of user, system, and nice are counted as active CPU use. *idle*, and *iowait* are considered as unused CPU capacity. These values start increasing from system start.

The averages over the last three intervals are taken and divided by the corresponding time interval. The resulting values are stored in the variables *CP_ActiveAVG* and *CP_idleAVG*. The corresponding rules are as follows:

```
HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"
```

The values of *CP_ActiveAVG* and *CP_idleAVG* are between 0 and 1. Therefore, 1 - *CP_ActiveAVG* is the unused CPU capacity. When multiplied by the number of active CPUs, it is specified in CPUs. When the total unused CPU capacity falls below 8% of a single CPU, a new CPU is added. If the total amount of idle capacity is larger than 115% (this is 15% more than one CPU free), a CPU is withdrawn.

The steal time is not included in these calculations. Steal time limits the amount of available CPU capacity, thus ensuring that no additional CPUs are plugged in, as this would worsen the scenario. You could consider including the steal times in the unplug rule, whereupon significant steal times would cause CPUs to be removed, leading to reduction in pressure on the physical CPUs.

Figure 6 shows the effect of the CPU management rules over time when managed based on real CPU load:

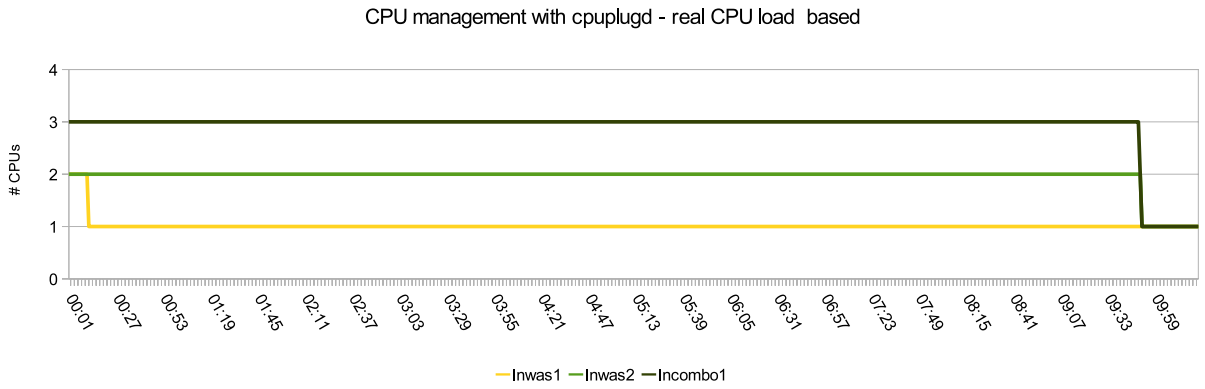

Figure 6. Number of active CPUs over time when managed by cpuplugd based on real CPU load

Table 9. Throughput and average CPU load when managed by cpuplugd based on real CPU load values

Configuration	TPS*	Relative CPU load*
real cpu load based	96%	96%
*100% is the manual-sized run		

Observation

The automated sizing values are the same as the manual sizing settings. The system reacts very fast to load variations. The throughput closely approximates the throughput of the manual sizing.

Conclusion

This is a very good solution when the objective is to adapt the number of CPUs directly to the load requirements. The automated values are very close to the manual settings. This rule is used for managing the number of active CPUs in all subsequent runs.

Memory plugging

These tests are aimed at providing the managed system with exactly the amount of memory required for an optimal system performance, thereby minimizing the system's memory usage.

The critical task is to detect that the system needs additional memory before the performance degrades too much, but not too early to limit the possible impact of memory overcommitment.

When an application requests additional storage, Linux memory management works as follows:

- If there are sufficient free pages, the request is served with no further actions.
- If that causes the amount of free memory to fall below a high water mark, an asynchronous page scan by kswapd is triggered in the background.
- If serving the request would cause the amount of free memory to fall below a low water mark, a so called direct scan is triggered, and the application waits until this scan provides the required pages.
- Depending on various other indicators the system may decide to mark anonymous pages (pages that are not related with files on disks) for swapping and initiate that these pages be written to swap asynchronously. After a memory page is backed up to disk it can be removed from memory. If it needs to be accessed later, it is retrieved from disk

From this, we conclude the following:

- The occurrence of page scans is a clear but soft indicator of memory pressure.
- The occurrence of direct page scans is an indicator of a serious lack of free memory pages likely to impact system performance, because applications are waiting for memory pages to free up.
- The amount of pages freed during the scans is reported as steal rate. The best case is a steal rate identical to the page scan rate, which would mean that each scanned page turns out to be a freeable page.
- The exact role of swapping with regard to the level of memory pressure is not clear at the moment, and it is therefore not considered in our test.

General considerations regarding cpuplugd rules for memory management

This section describes cpuplugd basic memory management, rule priority and how to calculate Linux guest sizes, as well as CMM pool sizing.

Rule priority

The cpuplugd mechanism ensures that the plugging rule (adding resources) always overrules the unplugging rule (removing resources), for both CPU and memory allocation. This protects the system against unexpected effects when testing overly aggressive unplugging rules.

Memory management basics

To identify memory which could be removed, any free pages could be taken as a first approach. However, a system continuously doing disk I/O, such as a database, will sooner or later use all its unused memory for page cache, so that no free memory remains. Therefore the memory used for *cache* and buffers need to be taken into account as well. The critical points here are:

- *Buffers* are used by the kernel, and a shortage here may lead to unpredictable effects
- *Page cache* is counted as *cache*
- *Shared memory* is also counted as *cache*

Considering *shared memory* as free memory is very critical for a system with a Java heap or database buffer pools, because they reside in shared memory. Therefore another approach is to calculate the page cache as the difference between cache and shared memory and consider this as free memory. The page cache itself always uses the oldest memory pages for new I/O requests or, in case of cache hits , the accessed page is marked as recently referenced.

Reducing the memory size leads to a reduction of the page cache at the cost of the oldest referenced pages. How much page cache is needed depends on the application type; web server and WebSphere have relatively low requirements, because in the case under study they are doing only a small amount of disk I/O, as the database itself constitutes a very powerful caching system.

The page scan rate is calculated as the sum of the following parameters:

- **vmstat.pgscan_kswapd_dma**
- **vmstat.pgscan_kswapd_normal**
- **vmstat.pgscan_kswapd_movable**

The direct page scan rate is calculated as the sum of the following parameters:

- **vmstat.pgscan_direct_dma**
- **vmstat.pgscan_direct_normal**
- **vmstat.pgscan_direct_movable**

The available part of the cache (from here on referred to as 'page cache') is calculated as the following difference:

`meminfo.Cached -meminfo.Shmem.`

All runs were done with a minimum cmm pool size of 0 (`CMM_MIN="0"`) and a maximum of the system size (5 GB) minus 256 KiB (`CMM_MAX="1245184"`) to allow the cmm pool to grow to the maximal possible size. Reserving 256 KiB for the kernel was intended as a safety net; if our rules work well the size of the cmm pool should never approach that maximum value .

Monitoring the guest sizes

The following methods were used to calculate the guest size during the tests:

- **Linux view:** the guest definition size minus the cmm pools size over time.
- **z/VM view:** sum of resident pages below 2 GB and pages above 2 GB from the UPAGE report (FCX113) over time.

These two views typically differ. One reason is that Linux provides a view on virtual memory allocation, while z/VM shows the physical memory allocation. Due to optimizations inside z/VM, not all virtual memory pages Linux allocates are backed up with physical memory.

CMM pool increments and decrements

Another important consideration is the size of the increments and decrements of the CMM pool. There are two important requirements:

- Starting an application, middleware or workload leads to a relatively large memory requirement in a short time. If the system does not react fast enough this in turn may lead to an out of memory exception. Our results indicate that the middleware typically does not allocate the entire amount of configured memory in a single step. For example, on the WebSphere Application Server systems with a 1 GB Java heap the CMM pool is reduced in multiple (4) steps when the workload was started, where only the first step was at the maximum size of 500 MiB. This softens this requirement for a fast reaction time, because even a system with a much larger Java heap (for example, 6 GB) would probably not require a `CMM_DEC` of 6 GB.
- A highly frequent oscillating CMM pool size should be avoided, because of the related overhead for operating system and z/VM.

These considerations suggest the following approach:

- The parameter **`CMM_INC`** was defined as the percentage of memory that is free (for example, 10%). This causes the increment of the pool to become smaller and smaller the closer the system comes to the 'ideal' configuration.
- The parameter **`CMM_DEC`** was defined as a percentage of the system size (for example, 10%, which would correspond to ~500 MiB). This results in a fix value independent of the current load situation and leads to a relatively fast decrement of the pool whenever a memory shortage is detected, depending on the applied rule.

The effect is an asymptotic increase of the CMM pool, up to the level where no more volatile memory is available for removal, while a request for new memory can be served in a small number of steps. With this setup both requirements where fulfilled in most cases.

CMM_MIN and CMM_MAX

The minimum size of the pool was specified as 0 pages, to allow full removal of the pool. As maximum value a very large value of 1,245,184 pages (4,864 MiB) was specified, which stops increasing the pool when less than 256 GB memory remains. The expectation was that the indicators for memory shortage would appear before the pool reaches that size, causing a reduction in the pool size, which in turn increases the available memory. This approach worked very well.

Optimizing for throughput

This topic briefly describes how the variables mentioned in the listings in this section are calculated.

For complete configuration file listings, refer to Appendix B, “cpuplugd configuration files,” on page 53.

The first test series was aimed at reaching a throughput close to the manual sized case. The following rules are analyzed:

Memory configuration 1 (plug: page scan, unplug: free + cache memory). The plugging rules are:

- `MEMPLUG="pgscanrate > 20" # kswapd + direct scans`

- **MEMUNPLUG="(meminfo.MemFree > meminfo.MemTotal / 10) | (cache > meminfo.MemTotal / 2)"**

  Memory is increased if the page scan rate (normal and direct page scans) exceeds 20 pages/sec. Memory is reduced if more than 10% of the total memory is free or if memory of the types cache and buffers exceeds 50% of the total memory. The rules use the values the variables have during the current interval.

The CMM pool increments are defined as:

- `CMM_INC="(meminfo.MemFree + cache) / 40"`
- `CMM_DEC="meminfo.MemTotal / 40"`

where cache means the memory reported as *cache* and as *buffers* in `/proc/meminfo`.

Memory configuration 2 (plug: page scan, unplug: free memory). The plugging rules are:

- `MEMPLUG="pgscanrate > 20" # kswapd + direct scans`
- **MEMUNPLUG="meminfo.MemFree > meminfo.MemTotal / 10 "**

  Memory is increased if the page scan rate (normal and direct page scans) exceeds 20 pages/sec. Memory is reduced if more than 10% of the total memory is free. The rules use the values the variables have during the current interval.

The CMM pool increments are defined as:

- `CMM_INC="meminfo.MemFree  / 40"`
- `CMM_DEC="meminfo.MemTotal / 40"`

Memory configuration 3 (plug: page scan, unplug: free memory + page cache). The plugging rules are:

- `MEMPLUG="pgscanrate > 20" # kswapd + direct scans`
- **MEMUNPLUG="(meminfo.MemFree + avail_cache) > (meminfo.MemTotal / 10)"**

  Memory is increased if the page scan rate (normal and direct page scans) exceeds 20 pages/sec. Memory is reduced if the sum of free memory and page cache (*avail_cache=cache- shared memory*) exceeds 10% of the total memory. The rules use the values the variables have during the current interval.

The CMM pool increments are defined as:

- `CMM_INC="meminfo.MemFree  / 40"`
- `CMM_DEC="meminfo.MemTotal / 40"`

Memory configuration 4 (plug: direct scan, unplug: free memory + page cache). The plugging rules are:

- **MEMPLUG="pgscanrate > 20" # direct scans only!**
- `MEMUNPLUG="(meminfo.MemFree + avail_cache) > ( meminfo.MemTotal / 10)"`

Memory is increased if the direct page scan rate exceeds 20 pages per second. Memory is reduced if the sum of free memory the page cache exceeds 10% of the total memory. The rules use the values the variables have during the current interval.

The CMM pool increments are defined as:

- `CMM_INC="meminfo.MemFree  / 40"`
- `CMM_DEC="meminfo.MemTotal / 40"`

Memory configuration 5 (plug: page scan vs steal, unplug: free memory + page cache). The plugging rules are:

- `MEMPLUG="pgscanrate > pgstealrate"  # kswapd + direct scans`
- `MEMUNPLUG="(meminfo.MemFree + avail_cache) > ( meminfo.MemTotal / 10)"`

Memory is increased if the page scan rate exceeds the page steal rate. Memory is reduced if the sum of free memory and page cache exceeds 10% of the total memory. The page scan rate exceeding the page steal rate indicates high memory usage while not all unused pages are consolidated at the end of the lists used for memory management. The strength of this rule is limited, and lies between normal pages scans a direct page scans. The system is no longer relaxed. The rules use the values the variables have during the last two intervals.

The CMM pool increments are defined as:

- `CMM_INC="meminfo.MemFree  / 40"`
- `CMM_DEC="meminfo.MemTotal / 40"`

Table 10 shows the result for these configurations for throughput and guest size:

Table 10. Impact of the various configuration rules for throughput and guest size

	Increase memory, if		Shrink memory, if				Guest size [MiB]*	
Configuration	Parameter	Rate [pages/sec]	Memory type	% of total memory	Relative TPS*	Relative LPAR CPU load*	Linux	z/VM
1	page scans	> 20	free (cache + buffers)	> 10% or > 50%	97%	99%	132%	115%
2	page scans	> 20	free	> 10%	97%	98%	131%	107%
3	page scans	> 20	free + page cache	> 10%	96%	99%	120%	114%
4	direct scans	> 20	free + page cache	> 10%	96%	99%	109%	105%
5	page scans	> page steal	free + page cache	> 10%	95%	97%	118%	105%
*100% is the manual sized run					higher is better	lower is better	closer to 100% is better	

Observation

The CPU load varies only slightly between scenarios. It is slightly lower than the manually sized run, but follows the throughput. The throughput also only varies slightly. Configuration 5 provides the lowest throughput and therefore the lowest CPU load. The resulting memory sizes are higher than the manually sized run.

Configurations 1 to 3 vary only the UNPLUG rule. It seems that the rule which uses page cache and free memory as a parameter to determine whether memory can be reduced (configuration 3) provides the smallest memory size at relatively high throughput values. In runs using the number of direct pages scans instead of kswapd page scans the system size is reduced further without additional impact on throughput or CPU load.

The VM view, which represents the real allocation of physical memory, typically shows lower values, which are much closer together than the Linux memory sizes.

Conclusion

The combination of using direct page scan rates to increase memory and using free memory and page cache to reduce memory is very suitable for memory management. It provides a throughput and memory size very close to the manually sized configuration. Interestingly, the plug and the unplug rules influence the system size. The expectation was that the plug rule would have no effect unless the system load changes.

It is expected that the smallest system results from using the direct scan rate instead of kswapd page scans for plugging memory, because direct scans are an indicator for a higher memory pressure. meaning the system tolerates a higher memory pressure before increasing memory.

Configuration 4 impacts throughput only slightly (-4%), but results in a memory size that is 9% larger than the manually sized configuration. This finding indicates that it is likely to be difficult to optimize both throughput and memory size at the same time.

More details about Linux memory size and throughput:

To understand better what happens in the various scenarios we compare the guest memory behavior for the rule with the largest guests (configuration 1) with the rule with the smallest guest (configuration 4).

Linux memory size for individual guests and three different configuration files

Figure 7 on page 33 shows the Linux memory size for the individual guests relative to the manually-sized configuration:

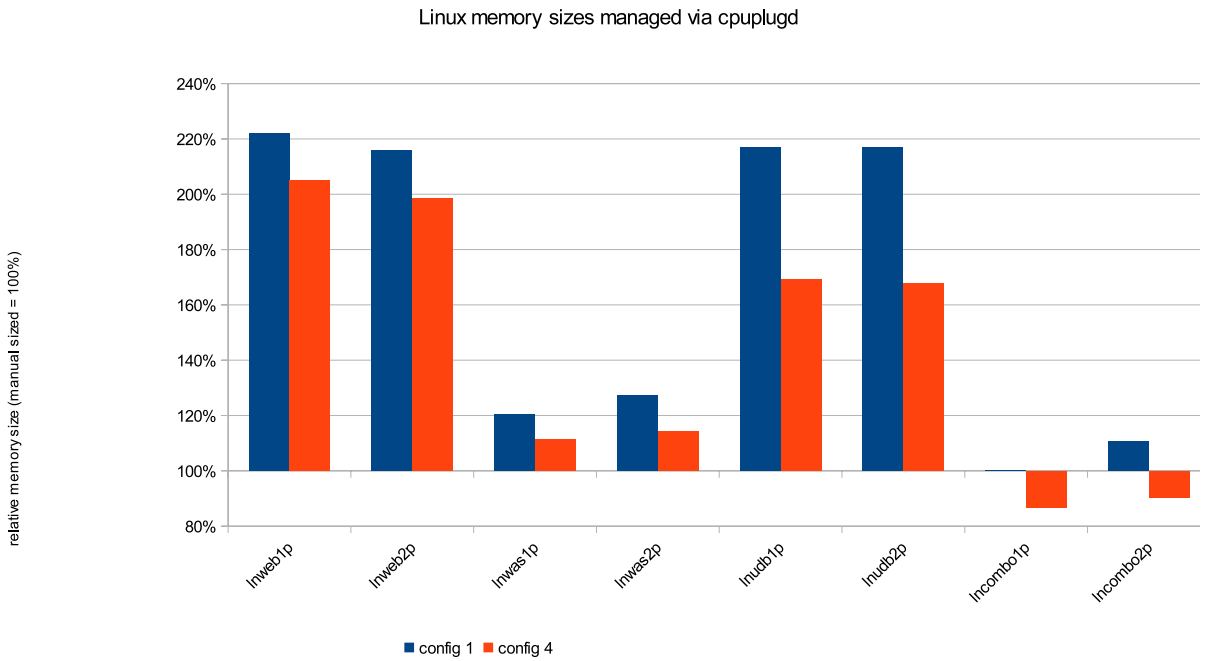

Figure 7. Relative Linux memory size for the individual guests for two different cpuplugd configuration files (manual sized = 100%)

Observation

The automated sizing of memory of the web server systems shows the worst results. None of our configurations allocates less than twice the manually sized memory. The database systems behave similar, but the rule with the direct scans allocates only around 50% too much memory. The sizing for the WebSphere systems is very good, especially when direct scans are used. Using this rule, the combos are even smaller than the manually sized systems.

Conclusion

The reason for oversizing the web servers by this much is most certainly caused by the small size of these systems when sized manually (342 MiB). The same argument applies to the database servers. Applying stronger conditions, especially with regard to the lower limits, will probably result in a better sizing. For a pure WebSphere system the direct scan rule is a very good fit.

Throughput reached for the individual components and three different configuration files

The fact that some of the combos are smaller than when manually sized, immediately raises the question whether the systems are too small. This should be evident from inspecting the reached throughput in Figure 8 on page 34:

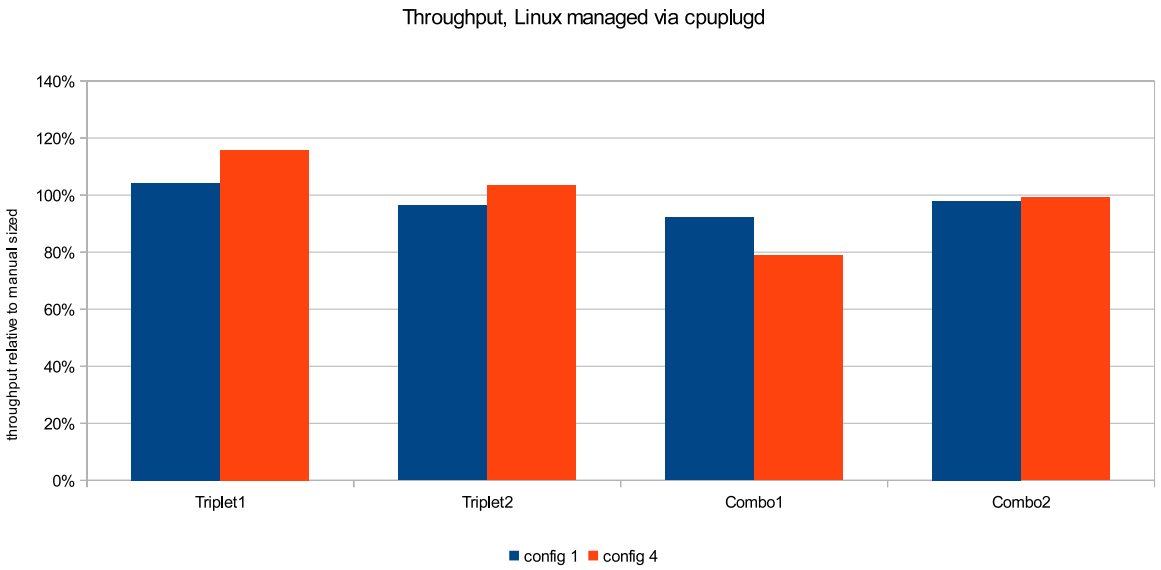

Figure 8. Throughput reached for the individual components and three different configuration files.

Observation

For both triplets, applying the direct scan rule leads to similar or even higher throughput than applying the manually sized configuration. Throughput for Combo2 is comparable to the throughput for the manually sized configuration while throughput for Combo1 is lower, especially when using the direct page scan rule.

Conclusion

The setup is relatively sensitive to memory sizes. The reason for the lower throughput for Combo1 is shown in the log output from cpulogd: the CPU plugging rule using direct scan provides only two CPUs for this system, where in the other scenarios Combo1 is allocated three CPUs. This confirms the impression that it will be difficult to optimize throughput and memory size with the same set of rules. It might be important to mention that the CPU cost spent to drive a certain throughput with these workload is very similar, even with the variations in throughput. That means that there is no overhead related to that.

CMM pool size

Looking at the size of the CMM pool over time shows that the same server types always behave in similar manner, even when the load on the triplets is different. The exception here are the combos, see Figure 9 on page 35 for an example:

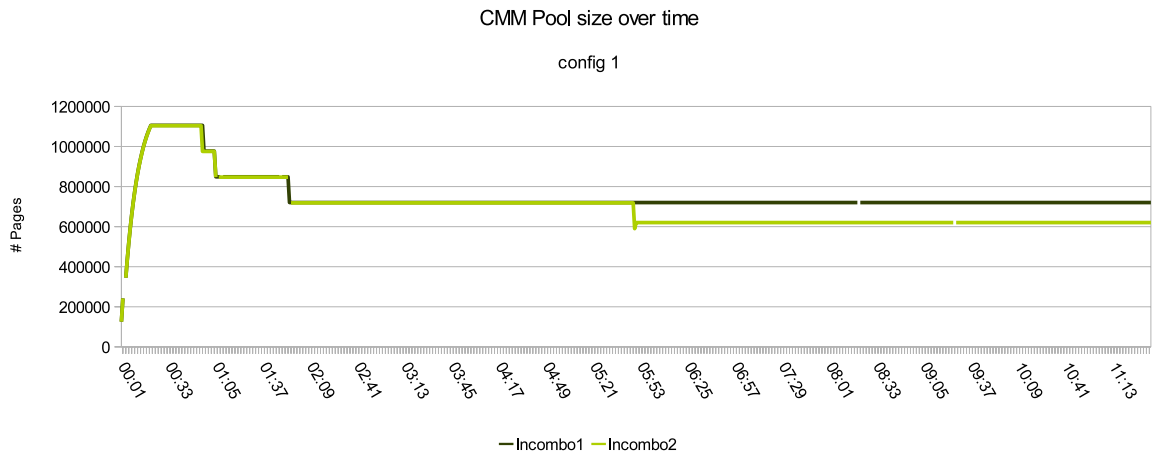

Figure 9. CMM Pool size over time with configuration 1

The other interesting topic is the impact of the rules on the memory sizing. Figures Figure 10, Figure 11 on page 36 and Figure 12 on page 36 show the cmm pool sizes over time for lnwas1, lnudb1 and lncombo2, for the rules with the largest memory sizes (configuration 1), and for the rules with the lowest memory sizes (configuration 4).

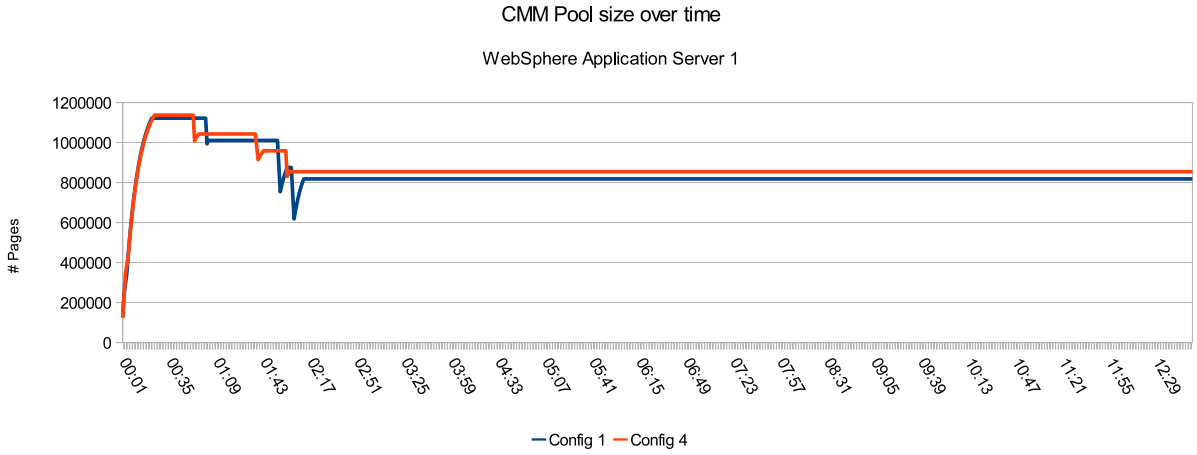

Figure 10. CMM Pool size over time with WebSphere Application Server 1

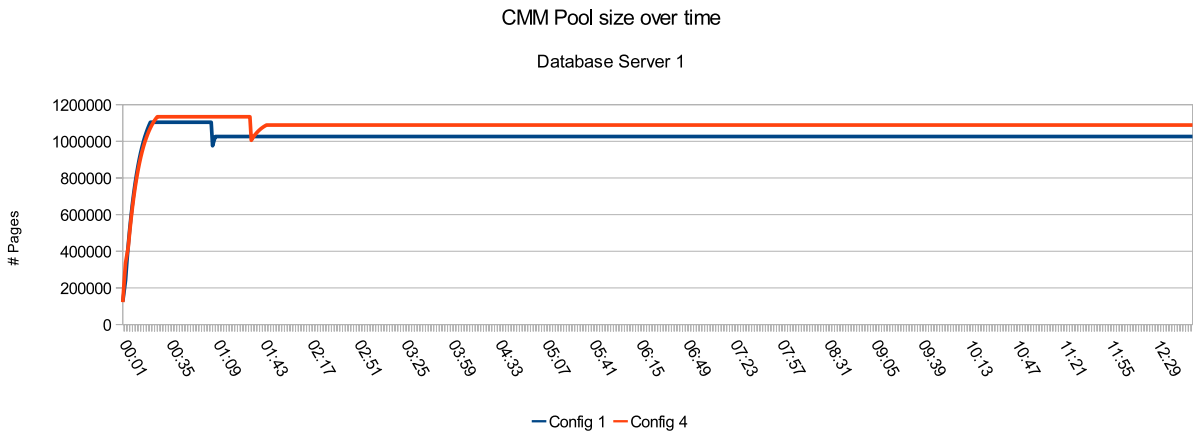

Figure 11. CMM Pool size over time for a database system

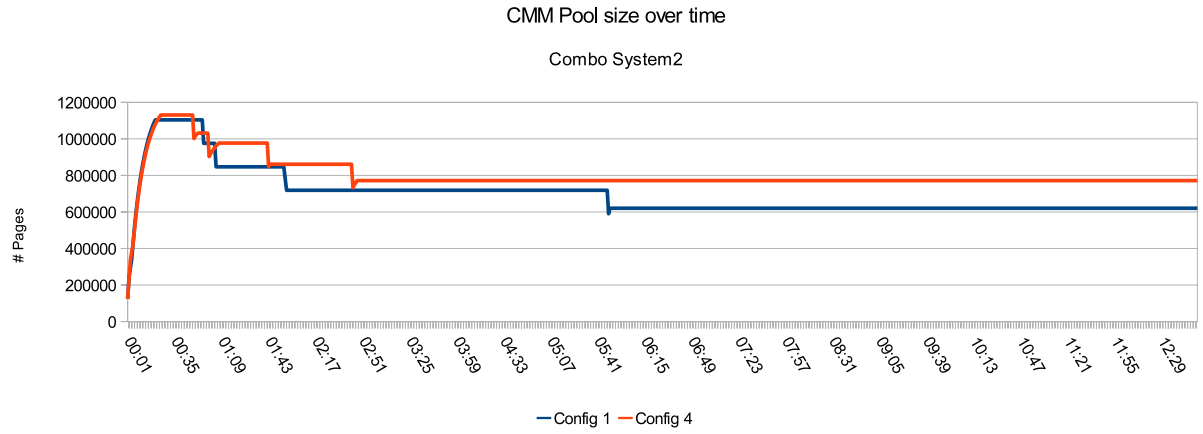

Figure 12. CMM Pool size over time for a Combo System (IHS, WebSphere Application Server, DB2)

Observation

In all scenarios we observe how first the CMM pool increases (meaning the guest systems yield memory to the Hypervisor) when the cpuplugd daemon is started. After 30 seconds the middleware servers are started, and after 60 seconds the workload is started and left to run for 10 minutes. The size of the CMM pool of all systems is relatively constant during the workload phase.

The largest difference between configuration 4 and configuration 1 results for the combos, the next largest difference results for the database systems and the smallest difference results for the WebSphere systems.

Conclusion

All configurations are very stable and react quickly to changing requirements. There are only small overswings where the pool was reduced by a large amount and then increased again. The configurations using direct page scans react slower and with smaller pool decreases than the configurations which also include

kswapd page scans. In the light of these results the latter configuration was not further evaluated.

Minimizing memory size

This text briefly describes how the variables mentioned in the listings in this section are calculated

For complete configuration file listings, refer to Appendix B, "cpuplugd configuration files," on page 53.

The second series of tests was aimed at minimizing the memory size further in order to reach the manually sized setup. The following rules are evaluated:

Memory configuration 7 (same as configuration 4, but with reduced free memory limit). The plugging rules are as follows:

- `MEMPLUG="pgscanrate > 20" # direct scans only!`
- `MEMUNPLUG="(meminfo.MemFree + avail_cache) > ( meminfo.MemTotal / 20)"`

Memory is increased if direct page scan rates exceed 20 pages/sec. Memory is reduced if the sum of free memory and page cache exceeds 5% of the total memory. The rules use the values the variables assumed during the current interval.

The CMM pool increments are defined as follows:

- `CMM_INC="meminfo.MemFree  / 40"`
- `CMM_DEC="meminfo.MemTotal / 40"`

Memory configuration 8 (same as configuration 7 with include page cache for CMM increment). The plugging rules are:

- `MEMPLUG="pgscanrate > 20" # direct scans only!`
- `MEMUNPLUG="(meminfo.MemFree + avail_cache) > ( meminfo.MemTotal / 20)"`

Memory is increased if direct page scan rates exceed 20 pages/sec. Memory is reduced if the sum of free memory and page cache exceeds 5% of the total memory. The rules use the values the variables assumed during the current interval .

The CMM pool increments are defined as follows:

- `CMM_INC="(meminfo.MemFree + avail_cache)  / 40"`
- `CMM_DEC="meminfo.MemTotal / 40"`

`CMM_INC`, which defines the chunk size when the CMM pool is increased, now includes the page cache, which should result in larger increments. All other scenarios have `CMM_INC="meminfo.MemFree / 40"`.

Table 11. Impact of the various configuration rules for throughput and guest size

	Increase memory if		Shrink memory if				Guest size (MiB)*	
Config	Parameter	Rate [pages/ sec]	Memory type	% of total memory	Relative TPS*	Relative LPAR CPU load*	Linux	z/VM
4	direct scans	> 20	free + page cache	> 10%	96%	99%	109%	105%

Table 11. Impact of the various configuration rules for throughput and guest size (continued)

Config	Increase memory if		Shrink memory if		Relative TPS*	Relative LPAR CPU load*	Guest size (MiB)*	
	Parameter	Rate [pages/ sec]	Memory type	% of total memory			Linux	z/VM
7	direct scans	> 20	free + page cache	> 5%	93%	98%	97%	99%
8	direct scans	> 20	free + page cache	> 5%	94%	97%	97%	99%
			`CMM_INC=(free+page cache) /40`					
*100% is the manual sized run					higher is better	lower is better	closer to 100% is better	

Observation

The total memory size is now smaller than the manually sized configuration, but the throughput is also lower.

Conclusion

There is a very slight advantage to using configuration 8 (the configuration with larger increments) over configuration 7, but the difference between them is very small.

More details about Linux memory size and throughput:

This topic describes the impact of Linux memory size and throughput for individual guests and different configuration files.

Impact of Linux memory size for the individual guests and three different configuration files

Figure 13 on page 39 shows the impact of the rule sets on the memory size of the individual servers:

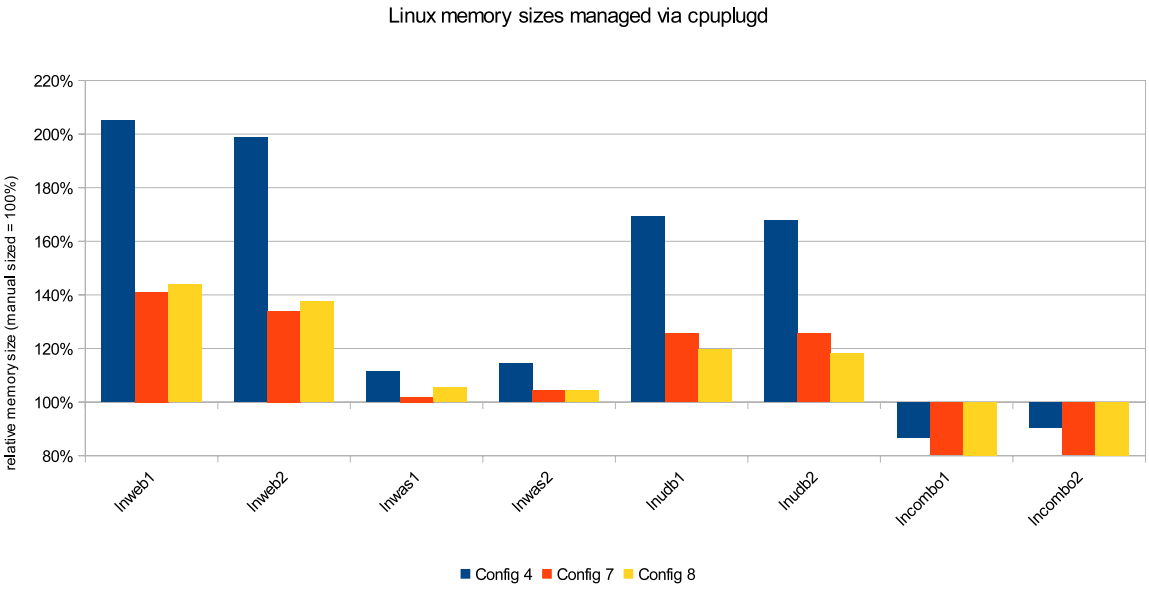

Figure 13. Relative Linux memory size for the individual guests and different configuration files. (manual sized = 100%)

Observation

In addition to the WebSphere Application Server, the database servers are now also very close to the manually sized systems. Even the web servers are only moderately oversized. The combos are further reduced in size.

Conclusion

It seems that configuration 7 is more appropriate for the Web and the Application Servers, while the database server is more optimally sized by configuration 8. When considering that the database servers are the only systems in our setup using a significant amount of page cache for disk I/O, this confirms that treating page cache as free memory for this purpose is a good approach. Remembering that configuration 4 leads to a throughput degradation for combos, it is to be expected that the rules evaluated here will perform even worse for the combos.

Throughput reached for the individual components and three different configuration files

The fact that some of the combos are smaller than when manually sized, immediately raises the question whether the systems are too small. This should be evident from inspecting the reached throughput in Figure 14 on page 40:

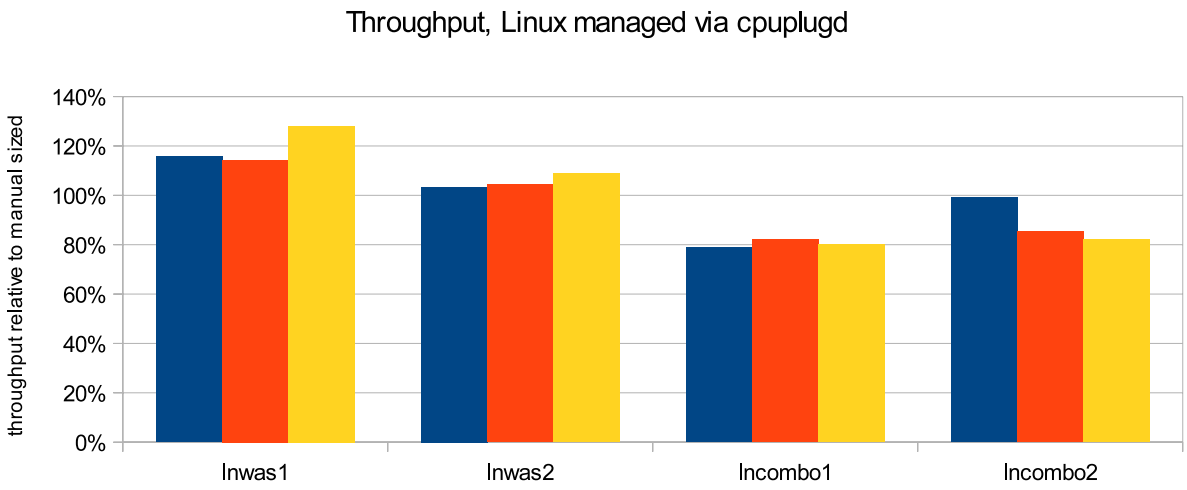

Figure 14. Throughput reached for the individual components and three different configuration files

Observation

The triplets achieve a higher throughput when applying these rules. The combos, however, suffer significantly.

Conclusion

It seems that the concept of using one set of rules to manage all servers is flawed by the issue that throughput and size can not be optimized at the same time. The rule sets using direct page scans for memory plugging and those using the sum of free memory and page cache (as computed from the difference between cache and shared memory) both perform well; the difference between them is which values are used as limits. It seems that compared to larger systems, smaller systems end up closer to the manually sized configuration when less memory is left free.

There are two approaches to select cpuplugd configuration files:

- A generic approach (which is our suggested default), which provides a good fit for all of our tested workloads and server types. It provides a slightly worse throughput, and slightly oversized systems (which leaves some space for optimizations for z/VM):
  - Plug memory when direct page scans exceed 20 pages/sec
  - `CMM DEC=total mem /40`
- A server type dependent approach:

Table 12. Recommended rules set depending on server type

	Server type	Recommended rules	CMM_INC	Unplug, when
1	Web server	configuration 7	free memory /40	(free mem+page cache) > 5%
2	WebSphere Application Server	configuration 7	free memory /40	(free mem+page cache) > 5%
3	Database server	configuration 8	(free mem+page cache)/40	(free mem+page cache) > 5%

Table 12. Recommended rules set depending on server type (continued)

	Server type	Recommended rules	CMM_INC	Unplug, when
4	Combo	configuration 4	free memory /40	(free mem+page cache) > 10%

Additional consideration:

- The Web servers, which are a front end for the Application servers, are very small systems. In this case it may be appropriate to use even smaller unplugging conditions. Important for these systems is that they just transfer the requests to the Application Server. A stand-alone Web server which provides a large amount of data from its file system is probably better treated in the same way as a database server.
- The alternative manual sizing needs to be done for each system individually and fits really well for one level of workload. But is also an valuable option when the additional win in performance is necessary, especially for servers with very constant resource requirements.

Dynamic runs

The tests in the previous sections are intended to determine the impact of the various rules. The next important question is how a system managed by cpuplugd reacts to load shifts.

For this purpose an additional group of guests are created (two additional triplets and two additional combo servers). The first group of guest are referred to as "Set 1", the new group of guests referred to as "Set 1" (see also "WebSphere environment" on page 15.

The steps in the experiment are as follows:

1. **load phase 1: Load on guest set 1**
   a. Start the middleware on the servers in guest set 1.
   b. Run the workload against the servers in guest set 1.
   c. Stop the workload and shut down the middleware running on the servers in guest set 1.
2. **wait phase 1**
   - Now that all guests in set 1 are warmed up, no middleware server or load is running, the important question is whether the guests release resources
   - The second set of guests are idle, and require few resources
3. **load phase 2: Load on guest set 2**
   a. Start the middleware on the servers in guest set 2.
   b. Run the workload against the servers in guest set 2.
   c. Stop the workload and shut down the middleware running on the servers in guest set 2.
4. **wait phase 2**
   - Now that all guests in all sets are warmed up, no middleware server or load is running
   - Resources allocated to servers in guest set 2 should be released
   - The question is whether the resource utilization reaches the same level it reached in wait phase 1
5. **load phase 3: Load on guest set 1**

a. Start the middleware on the servers in guest set 1.
b. Run the workload against the servers in guest set 1.
c. Stop the workload and shut down the middleware running on the servers in guest set 1.

The cpuplugd configuration used is configuration 2:

Memory configuration 2 (page scan, free memory). The plugging rules are:

- `MEMPLUG="pgscanrate > 20" # kswapd + direct scans`
- `MEMUNPLUG="meminfo.MemFree > meminfo.MemTotal / 10 "`

Memory is increased if the page scan rate exceeds 20 pages per second. Memory is reduced if more than 10% of the total memory is free. The rules use the values the variables have during each interval.

The CMM pool increments are defined as follows:

- `CMM_INC="meminfo.MemFree  / 40"`
- `CMM_DEC="meminfo.MemTotal / 40"`

Figure 15 shows the amount of free memory in z/VM over time for the manually sized configuration and for cpuplugd with configuration 2 as reported from the z/VM performance toolkit report AVAILOG (FCX254).

In the manually sized case, the total memory size from the guests of one set is 9,508 MiB, both sets together are defined with 19,016 MiB. The z/VM system size is 20 GB.

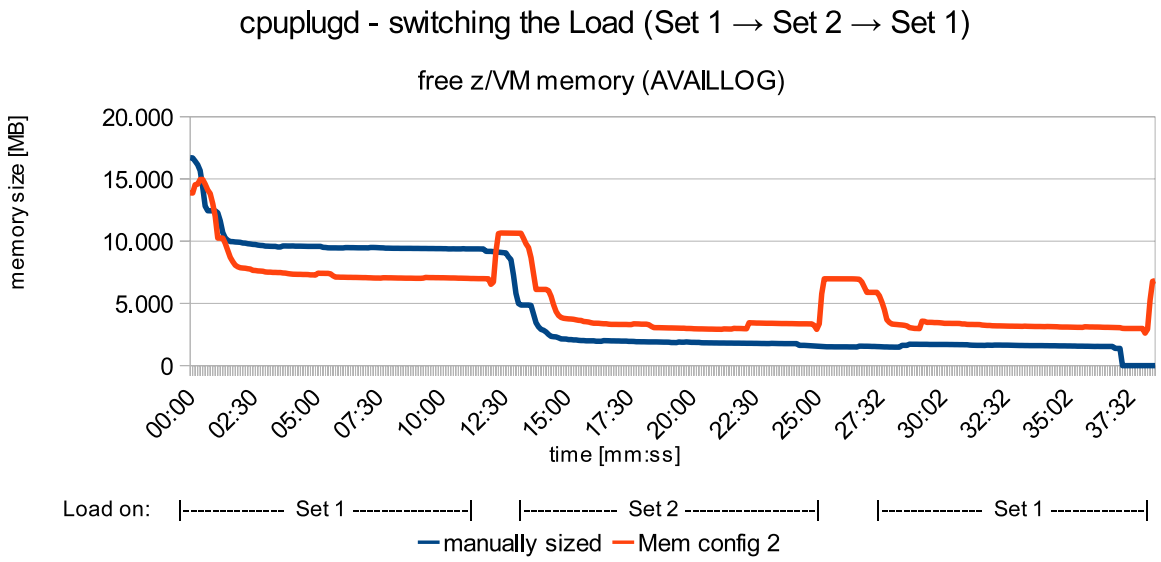

Figure 15. Free z/VM memory over time when switching the workload from guest set 1 to guest set 2.

Observations

The free memory during load phase 1 is larger in the manually sized scenarios than in the cpuplugd managed scenario. In both wait phases, when the middleware is shut down, some memory is given back in the cpuplugd managed

case, while the manually sized scenario shows no reaction on that. This means that the cpuplugd managed scenario frees up more memory and this does not change for the remainder of the test. Comparing the manual configuration with the system being managed by cpuplugd, the latter scenario uses up about 1.5 GB less memory.

When the middleware and load are started on guest set 2 in load phase 2, a significant amount of memory from these guests is allocated.

Conclusion

cpuplugd automatically adapts memory size to the requirements. This simplifies systems management significantly. In case of changing requirements (for example when one guest frees memory because an application terminates while another guest raises memory requirements because an application is started), the automated memory management resulted in a lower memory footprint as compared to the manually sized setup.

CMM pools

The fact that the memory size never comes back to the level of phase 1 merits a more detailed look at the CMM pools. Figure 16 shows the Linux memory size calculated as defined guest size (5 GB) – CMM pool for two selected guests the combo 2 from Set 1 and combo 4 from Set 2.

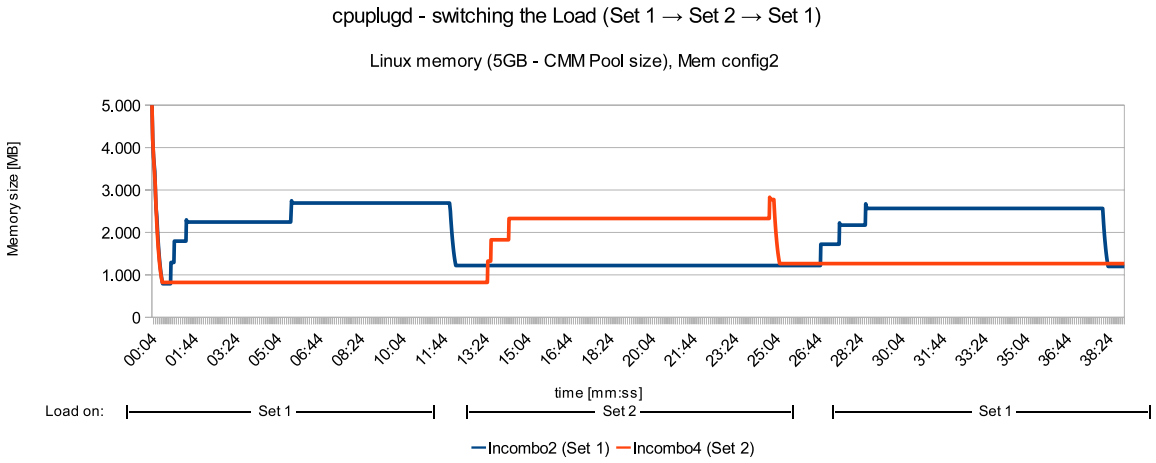

Figure 16. Linux memory size calculated as defined guest size (5 GB) – CMM pool size when switching the workload from guest set 1 (lncombo2)to guest set 2 (lncombo4)

Observations

The CMM pools grow and shrink according to the load and memory requirements of the servers. The memory sizes after the first load shift phases are very similar on both systems, and slightly higher than before.

Conclusion

The cpuplugd daemon works as expected. The fact that this is not reflected in the z/VM view is caused by a lack of real memory pressure in z/VM. There is no hard requirement from z/VM to take away the pages from the guests.

Compare the number of active guest CPUs

Figure 17 and Figure 18 compare the number of active guest CPUs over time for selected guests in the manually sized configuration with cpuplugd using configuration 2. The guests running web servers and databases, as well as the low-utilization WebSphere Application Server, always run with 1 CPU and are omitted. The behavior of the two combo system in each guest set is very similar, therefore only one combo is shown.

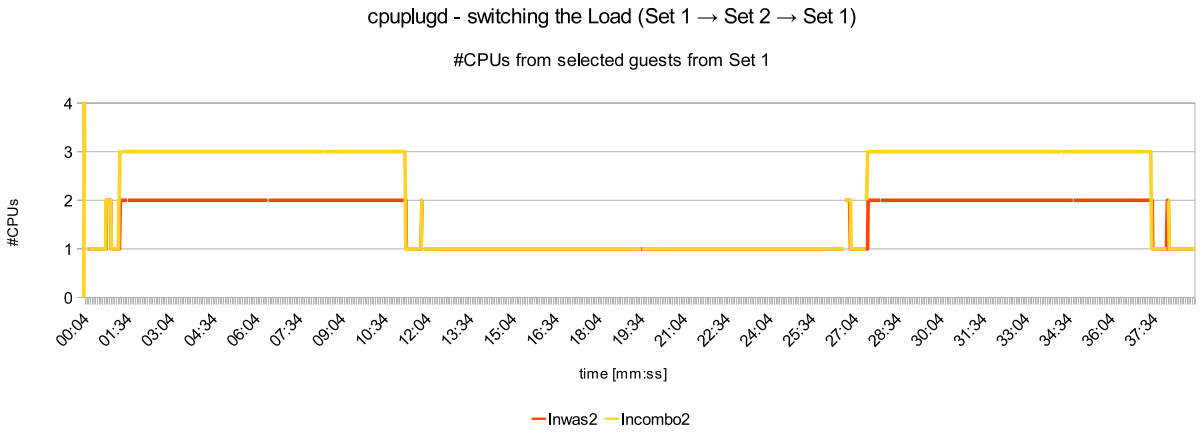

Figure 17. Number of active CPUs for a WebSphere guest and a Combo guest of Set 1 over time when switching the workload from guest set 1 to guest set 2

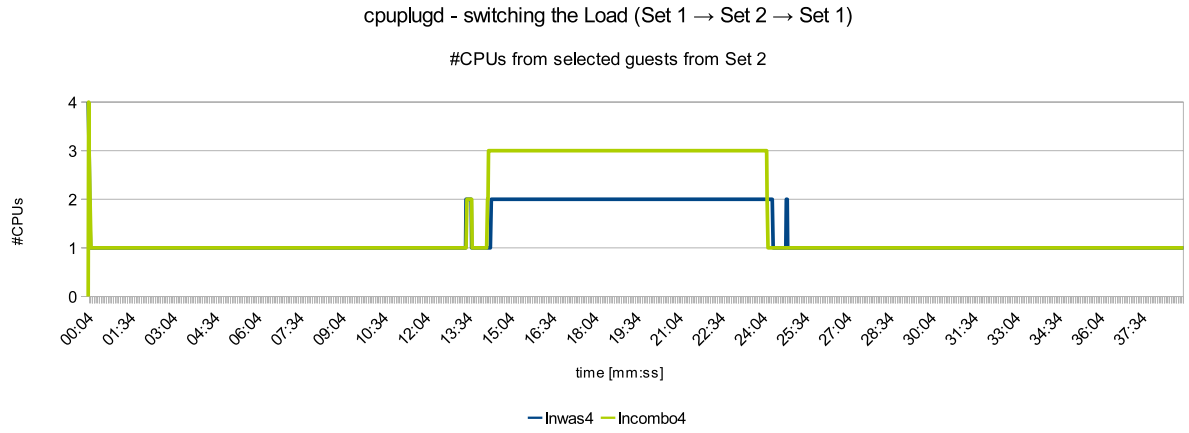

Figure 18. Number of active CPUs for a WebSphere guest and a Combo guest of Set 2 over time when switching the workload from guest set 1 to guest set 2

Observations

The number of CPUs exactly follows the workload load pattern. The small peaks at the start and the end are from starting and stopping the middleware. The increase from 1 to 3 CPUs happens over two intervals, adding one CPU in each step.

Conclusion

The chosen rules are very suitable to provide the system with the appropriate CPU resources while reacting very quickly to workload changes.

Setup tests and variations

Four setup tests are described along together with their results and conclusions

Scaling the cpuplugd update interval

This topic text briefly describes how the variables mentioned in the listings in this section are calculated.

For complete configuration file listings, refer to Appendix B, "cpuplugd configuration files," on page 53.

This series of test aims to compare the impact of different update intervals (parameter **UPDATE**). The following rules are evaluated:

Memory configuration 2 (page scan, free memory, update 1 sec). The **UPDATE** parameter is set to 1 second (default value used for this study). The plugging rules are:

- `MEMPLUG="pgscanrate > 20" # kswapd + direct scans`
- `MEMUNPLUG="meminfo.MemFree > meminfo.MemTotal / 10 "`

Memory is increased when the page scan rate exceeds 20 page per second. Memory is reduced when the amount of free memory exceeds 10% of total memory.

The CMM pool increments are defined as follows:

- `CMM_INC="meminfo.MemFree  / 40"`
- `CMM_DEC="meminfo.MemTotal / 40"`

Memory configuration 9 (page scan, free memory, update 2 sec). The **UPDATE** parameter is set to 2 seconds. The plugging rules are the same as those in configuration 2.

Memory configuration 10 (page scan, free memory, update 5 sec). The **UPDATE** parameter is set to 5 seconds. The plugging rules are the same as those in configuration 2, but they use only the values from the current interval, which covers now 5 seconds. The others runs use an average of the last three values.

Table 13 shows the results when scaling the cpuplugd **UPDATE** interval.

Table 13. Impact of scaling the cpuplugd UPDATE interval on throughput and guest size

Configuration	Update interval (seconds)	Increase memory, if	Shrink memory, if	Relative TPS*	Relative LPAR CPU load*	Guest size (MiB)*	
						Linux	z/VM
2	1	page scans > 20 pages/sec	Free > 10% of total memory	97%	98%	131%	107%
9	2			93%	99%	131%	119%
10	5			96%	96%	124%	109%
*100% is the manual sized run				higher is better	lower is better	closer to 100% is better	

Observation

The throughput for the scenario using an update interval of 2 seconds is lower than expected, but the throughput for the scenario using an update interval of 5 seconds is close to the throughput for the scenario using an update interval of 1 second. The CPU load shows no clear tendency either. The sum of the guest sizes in the Linux view decreases as the update interval is increased.

Conclusion

The run using an update interval of 5 seconds is consistent with the run using an update interval of 1 second in the sense that the guest size, throughput and CPU load decrease. The run using an update interval of 2 seconds seems to be affected by other unknown influences.

Determining whether cpuplugd activity depends on CPU load

Figure 19 is used to determine if cpuplugd activity depends on the CPU load. The figure shows the CPU cost per transaction for the manually sized run as a function of the duration of the cpuplugd **UPDATE** interval.

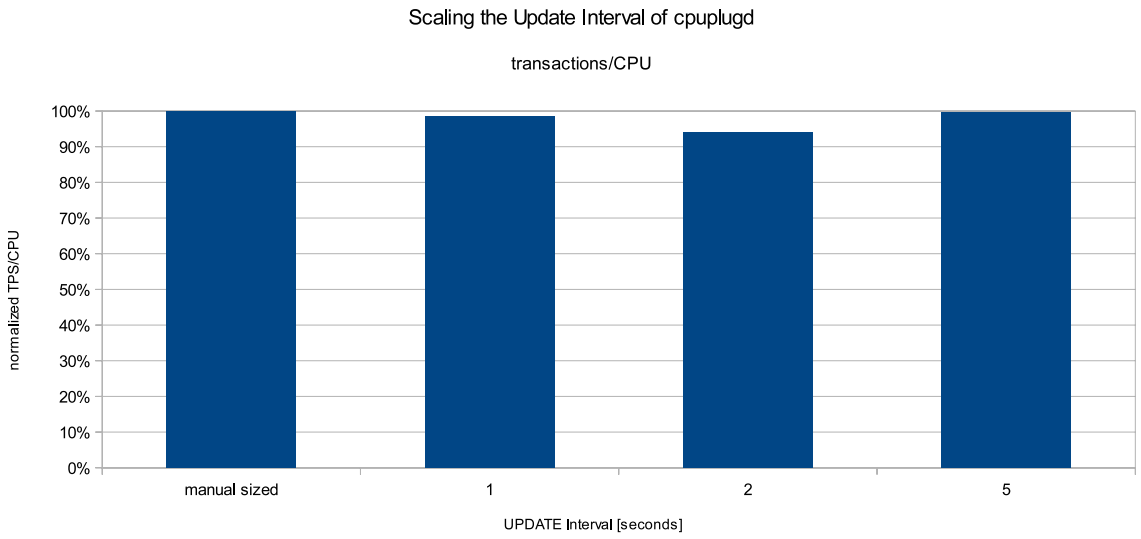

Figure 19. CPU cost per transaction for the manual sized run as a function of the duration of the cpuplugd UPDATE interval

Observation

The CPU cost per transaction for all scenarios is very similar, with the exception of the scenario using an update interval of 2 seconds, which yielded unexpected results.

Conclusion

The expectation was that the overhead caused by cpuplugd would increase as the update interval is made shorter. However, under normal workload conditions we see no differences, meaning that evaluating the rules seems to create no noteworthy additional CPU cost. If cpuplugd changes the system configuration

with a very high frequency (for example each interval) a different result may be obtained.

CMM pool size over time for scaling the cpuplugd UPDATE interval

Figure 20 is used to determine if cpuplugd activity depends on the CPU load. The figure shows the CPU cost per transaction for the manually sized run as a function of the duration of the cpuplugd **UPDATE** interval.

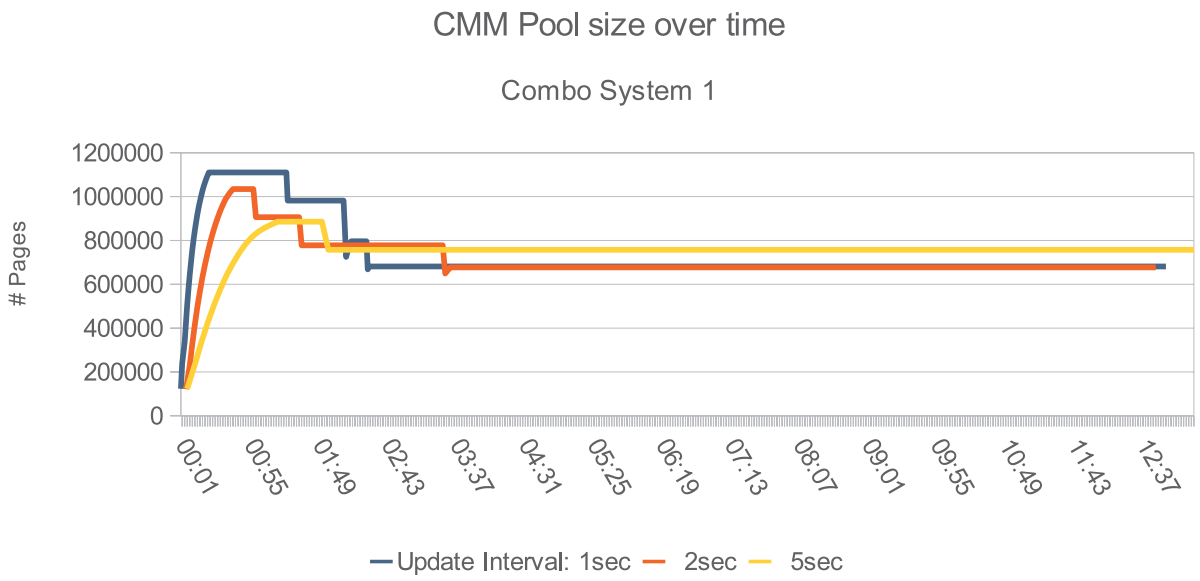

Figure 20. CMM pools size over time for scaling the cpuplugd UPDATE interval

Observation

The system reaction to load changes becomes more moderate as the update interval increases. When applying an update interval of 5 seconds the CMM pool stays smaller than for shorter intervals, but in steady state it remains larger than for the shorter intervals. This is true for the Combo systems, as well as for the WebSphere Application Server system with the higher load.

Conclusion

Neither cpuplugd activity nor the size of the **UPDATE** interval generates a significant overhead in terms of additional CPU cost. The **UPDATE** value can be used to determine how fast a system should react to changing requirements.

The recommended approach is to start with an update interval of 1 second and monitor the behavior of the system. If the update interval is changed too frequently, it is possible to calm down the system by increasing the update interval at the cost of a slower reaction to load changes. An alternative method to achieve a flattening effect is to average update values over several intervals.

Memory plugging and steal time

The cpuplugd tests with memory plugging revealed a serious problem when a kernel compile was used as workload.

The result is shown in Figure 21.

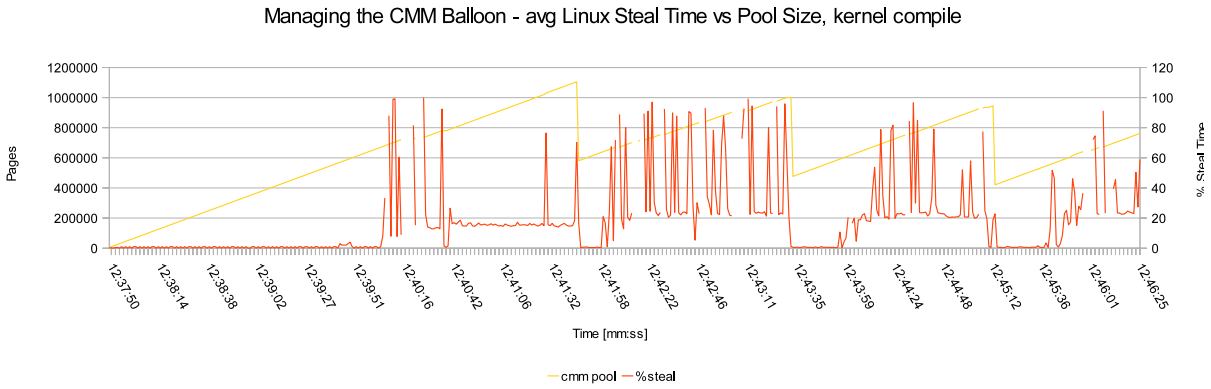

Figure 21. CMM Pools size and CPU steal time over time when compiling a Linux kernel

Observation

During the increase of the CMM pool the steal time frequently reaches values between 90% and 100%. The issue appears on z/VM 5.4 and on z/VM 6.1 when the APAR described below is not installed.

Conclusion

It is not recommended using cpuplugd to manage the memory size of a guest without the APAR installed, described in the next section.

Figure 22 shows the behavior after installing the fix released in APAR VM65060.

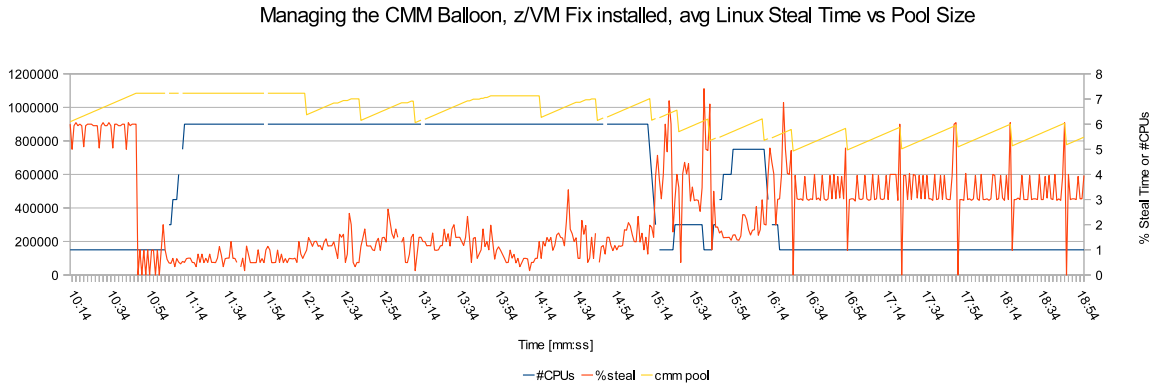

Figure 22. CMM Pools size and CPU steal time over time when compiling a Linux kernel with the fix released in APAR VM65060 installed

Observation

After installing the fix the steal time is between 1% and 4% for most of the time instead of 90%-100% without the fix, see Figure 21. The graph also shows that the number of assigned CPUs increases during the compile phase and decreases afterwards during the link phase, in line with the processing power required.

Conclusion

Installing the z/VM APAR VM65060 fix is required when memory management with cpuplugd is planned to avoid excessive steal time numbers. It is available for z/VM 5.4, z/VM 6.1, and z/VM 6.2.

Appendix A. Tuning scripts

These tuning scripts can be run.

DB2 UDB tuning

The following tuning script is run every time the DB2 database for trade, tradedb is created or recreated, and populated with test data.

```
db2 -v "connect to tradedb"
db2 -v "update db cfg for tradedb using DBHEAP 25000"
db2 -v "update db cfg for tradedb using CATALOGCACHE_SZ 282"
db2 -v "update db cfg for tradedb using LOGBUFSZ 8192"
db2 -v "update db cfg for tradedb using BUFFPAGE 366190"
db2 -v "update db cfg for tradedb using LOCKLIST 1000"
db2 -v "update db cfg for tradedb using SORTHEAP 642"
db2 -v "update db cfg for tradedb using STMTHEAP 2048"
db2 -v "update db cfg for tradedb using PCKCACHESZ 7500"
db2 -v "update db cfg for tradedb using MAXLOCKS 75"
db2 -v "update db cfg for tradedb using MAXAPPLS 500"
db2 -v "update db cfg for tradedb using LOGFILSIZ 5000"
db2 -v "update db cfg for tradedb using LOGPRIMARY 6"
db2 -v "update db cfg for tradedb using LOGSECOND 6"
db2 -v "update db cfg for tradedb using SOFTMAX 70"
db2 -v "update dbm cfg using MAXAGENTS 200"
db2 -v "update dbm cfg using NUM_POOLAGENTS -1"
db2 -v "update dbm cfg using MAX_QUERYDEGREE -1"
db2 -v "update dbm cfg using FCM_NUM_BUFFERS 512"
db2 -v "update dbm cfg using FCM_NUM_RQB 256"
db2 -v "update dbm cfg using DFT_MON_LOCK OFF"
db2 -v "update dbm cfg using DFT_MON_BUFPOOL ON"
db2 -v "update dbm cfg using DFT_MON_STMT OFF"
db2 -v "update dbm cfg using DFT_MON_TABLE OFF"
db2 -v "update dbm cfg using DFT_MON_UOW OFF"
db2 -v "alter bufferpool ibmdefaultbp size 500"
db2 -v "reorgchk update statistics on table all"
db2 -v "connect reset"
db2 -v "terminate"
```

WebSphere tuning script

The WebSphere tuning script is run only once because the effects are persistent.

The script is run as follows:

```
/opt/IBM/WebSphere/AppServer/bin/wsadmin.sh -f tuneDayTrader.py server server1
```

The values for the JVM heap are then overridden manually. These are common WebSphere tuning variables, which are set to values to optimize the performance of the DayTrader application.

The `tuneDayTrader.py` python script is provided in the downloaded DayTrader source zip file.

Appendix B. cpuplugd configuration files

Sample configuration scripts

Recommended default configuration

```
UPDATE="1"

CPU_MIN="1"
CPU_MAX="0"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"

user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"

CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"

CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"
CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"

pgscanrate="(pgscan_d - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
avail_cache="meminfo.Cached -meminfo.Shmem"

CMM_MIN="0"
CMM_MAX="1245184"

CMM_INC="meminfo.MemFree  / 40"
CMM_DEC="meminfo.MemTotal / 40"

MEMPLUG   = "pgscanrate > 20"
MEMUNPLUG = "(meminfo.MemFree + avail_cache) > (meminfo.MemTotal / 10)"
```

CPU plugging via loadavg

```
UPDATE="1"

CPU_MIN="1"
CPU_MAX="0"

HOTPLUG="(loadavg > onumcpus + 0.75) & (idle < 10.0)"
HOTUNPLUG="(loadavg < onumcpus - 0.25) | (idle > 50)"

CMM_MIN="0"
CMM_INC="0"
```

```
CMM_DEC="0"

MEMPLUG="0"
MEMUNPLUG="0"
```

CPU plugging via real CPU load

```
UPDATE="1"

CPU_MIN="1"
CPU_MAX="0"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"

user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"

CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"

CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

CMM_MIN="0"
CMM_INC="0"

CMM_DEC="0"

MEMPLUG="0"
MEMUNPLUG="0"
```

Memory plugging configuration 1

```
UPDATE="1"

CPU_MIN="1"
CPU_MAX="0"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"
user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"

CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
```

```
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"

CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

pgscan_k="vmstat.pgscan_kswapd_dma[0] + vmstat.pgscan_kswapd_normal[0] + vmstat.pgscan_kswapd_movable[0]"
pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_k1="vmstat.pgscan_kswapd_dma[1] + vmstat.pgscan_kswapd_normal[1] + vmstat.pgscan_kswapd_movable[1]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"
pgscanrate="(pgscan_k + pgscan_d - pgscan_k1 - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
cache="meminfo.Cached + meminfo.Buffers"

CMM_MIN="0"
CMM_MAX="1245184"

CMM_INC="(meminfo.MemFree + cache) / 40"
CMM_DEC="meminfo.MemTotal / 40"

MEMPLUG = "pgscanrate > 20"
MEMUNPLUG = "(meminfo.MemFree > meminfo.MemTotal / 10) | (cache > meminfo.MemTotal / 2)"
```

Memory plugging configuration 2

```
UPDATE="1"

CPU_MIN="1"
CPU_MAX="0"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"

user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"

CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"

CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

pgscan_k="vmstat.pgscan_kswapd_dma[0] + vmstat.pgscan_kswapd_normal[0] + vmstat.pgscan_kswapd_movable[0]"
pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_k1="vmstat.pgscan_kswapd_dma[1] + vmstat.pgscan_kswapd_normal[1] + vmstat.pgscan_kswapd_movable[1]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"
pgscanrate="(pgscan_k + pgscan_d - pgscan_k1 - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"

CMM_MIN="0"
CMM_MAX="1245184"

CMM_INC="meminfo.MemFree  / 40"
CMM_DEC="meminfo.MemTotal / 40"
```

```
MEMPLUG = "pgscanrate > 20"
MEMUNPLUG = "meminfo.MemFree > meminfo.MemTotal / 10 "
```

Memory plugging configuration 3

```
UPDATE="1"

CPU_MIN="1"
CPU_MAX="0"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"

user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"

CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"

CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

pgscan_k="vmstat.pgscan_kswapd_dma[0] + vmstat.pgscan_kswapd_normal[0] + vmstat.pgscan_kswapd_movable[0]"
pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_k1="vmstat.pgscan_kswapd_dma[1] + vmstat.pgscan_kswapd_normal[1] + vmstat.pgscan_kswapd_movable[1]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"
pgscanrate="(pgscan_k + pgscan_d - pgscan_k1 - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
avail_cache="meminfo.Cached -meminfo.Shmem"

CMM_MIN="0"
CMM_MAX="1245184"

CMM_INC="meminfo.MemFree  / 40"
CMM_DEC="meminfo.MemTotal / 40"

MEMPLUG="pgscanrate > 20"
MEMUNPLUG="(meminfo.MemFree + avail_cache) > ( meminfo.MemTotal / 10)"
```

Memory plugging configuration 4

```
UPDATE="1"

CPU_MIN="1"
CPU_MAX="0"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"

user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"
```

```
CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"

CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"
pgscanrate="(pgscan_d - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
avail_cache="meminfo.Cached -meminfo.Shmem"

CMM_MIN="0"
CMM_MAX="1245184"

CMM_INC="meminfo.MemFree  / 40"
CMM_DEC="meminfo.MemTotal / 40"

MEMPLUG = "pgscanrate > 20"
MEMUNPLUG =  "(meminfo.MemFree + avail_cache) > ( meminfo.MemTotal / 10)"
```

Memory plugging configuration 5

```
UPDATE="1"

CPU_MIN="1"
CPU_MAX="0"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"

user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"

CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"

CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

pgscan_k="vmstat.pgscan_kswapd_dma[0] + vmstat.pgscan_kswapd_normal[0] + vmstat.pgscan_kswapd_movable[0]"
pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_k2="vmstat.pgscan_kswapd_dma[2] + vmstat.pgscan_kswapd_normal[2] + vmstat.pgscan_kswapd_movable[2]"
pgscan_d2="vmstat.pgscan_direct_dma[2] + vmstat.pgscan_direct_normal[2] + vmstat.pgscan_direct_movable[2]"
pgscanrate="(pgscan_k + pgscan_d - pgscan_k2 - pgscan_d2)"
pgsteal="vmstat.pgsteal_dma + vmstat.pgsteal_normal + vmstat.kswapd_steal + vmstat.pgsteal_movable"
```

```
pgsteal2="vmstat.pgsteal_dma[2] + vmstat.pgsteal_normal[2] + vmstat.kswapd_steal[2] + vmstat.pgsteal_movable[2]"
pgstealrate="(pgsteal-pgsteal2)"

avail_cache="meminfo.Cached -meminfo.Shmem"

CMM_MIN="0"
CMM_MAX="1245184"

CMM_INC="meminfo.MemFree  / 40"
CMM_DEC="meminfo.MemTotal / 40"

MEMPLUG = "pgscanrate > pgstealrate"
MEMUNPLUG =  "(meminfo.MemFree + avail_cache) > ( meminfo.MemTotal / 10)"
```

Memory plugging configuration 7

```
UPDATE="1"

CPU_MIN="1"
CPU_MAX="0"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"

user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"

CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"

CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"
pgscanrate="(pgscan_d - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
avail_cache="meminfo.Cached -meminfo.Shmem"

CMM_MIN="0"
CMM_MAX="1245184"

CMM_INC="meminfo.MemFree  / 40"
CMM_DEC="meminfo.MemTotal / 40"

MEMPLUG = "pgscanrate > 20"
MEMUNPLUG =  "(meminfo.MemFree + avail_cache) > ( meminfo.MemTotal / 20)"
```

Memory plugging configuration 8

```
UPDATE="1"

CPU_MIN="1"
CPU_MAX="0"
```

```
user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"

user_2="(cpustat.user[2] - cpustat.user[3])"
nice_2="(cpustat.nice[2] - cpustat.nice[3])"
system_2="(cpustat.system[2] - cpustat.system[3])"

CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active2="(user_2 + nice_2 + system_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_ActiveAVG="(CP_Active0+CP_Active2) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
idle_2="(cpustat.idle[2] - cpustat.idle[3])"
iowait_2="(cpustat.iowait[2] - cpustat.iowait[3])"

CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle2="(idle_2 + iowait_2)/ (cpustat.total_ticks[2] - cpustat.total_ticks[3])"
CP_idleAVG="(CP_idle0 + CP_idle2) / 2"

HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"
pgscanrate="(pgscan_d - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
avail_cache="meminfo.Cached -meminfo.Shmem"

CMM_MIN="0"
CMM_MAX="1245184"

CMM_INC="(meminfo.MemFree + avail_cache)  / 40"
CMM_DEC="meminfo.MemTotal / 40"

MEMPLUG = "pgscanrate > 20"
MEMUNPLUG =  "(meminfo.MemFree + avail_cache) > ( meminfo.MemTotal / 20 )"
```

Memory plugging configuration 9

```
UPDATE="2"

CPU_MIN="1"
CPU_MAX="0"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"
user_1="(cpustat.user[1] - cpustat.user[2])"
nice_1="(cpustat.nice[1] - cpustat.nice[2])"
system_1="(cpustat.system[1] - cpustat.system[2])"

CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_Active1="(user_1 + nice_1 + system_1)/ (cpustat.total_ticks[1] - cpustat.total_ticks[2])"

CP_ActiveAVG="(CP_Active0+CP_Active1) / 2"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"

idle_1="(cpustat.idle[1] - cpustat.idle[2])"
iowait_1="(cpustat.iowait[1] - cpustat.iowait[2])"

CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idle1="(idle_1 + iowait_1)/ (cpustat.total_ticks[1] - cpustat.total_ticks[2])"
CP_idleAVG="(CP_idle0 + CP_idle1) / 2"
```

```
HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

pgscan_k="vmstat.pgscan_kswapd_dma[0] + vmstat.pgscan_kswapd_normal[0] + vmstat.pgscan_kswapd_movable[0]"
pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_k1="vmstat.pgscan_kswapd_dma[1] + vmstat.pgscan_kswapd_normal[1] + vmstat.pgscan_kswapd_movable[1]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"

pgscanrate="(pgscan_k + pgscan_d - pgscan_k1 - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"

CMM_MIN="0"
CMM_MAX="1245184"

CMM_INC="meminfo.MemFree  / 40"
CMM_DEC="meminfo.MemTotal / 40"

MEMPLUG = "pgscanrate > 20"
MEMUNPLUG = "meminfo.MemFree > meminfo.MemTotal / 10 "
```

Memory plugging configuration 10

```
UPDATE="5"

CPU_MIN="1"
CPU_MAX="0"

user_0="(cpustat.user[0] - cpustat.user[1])"
nice_0="(cpustat.nice[0] - cpustat.nice[1])"
system_0="(cpustat.system[0] - cpustat.system[1])"

CP_Active0="(user_0 + nice_0 + system_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_ActiveAVG="(CP_Active0)"

idle_0="(cpustat.idle[0] - cpustat.idle[1])"
iowait_0="(cpustat.iowait[0] - cpustat.iowait[1])"
CP_idle0="(idle_0 + iowait_0)/ (cpustat.total_ticks[0] - cpustat.total_ticks[1])"
CP_idleAVG="(CP_idle0)"

HOTPLUG="((1 - CP_ActiveAVG) * onumcpus) < 0.08"
HOTUNPLUG="(CP_idleAVG * onumcpus) > 1.15"

pgscan_k="vmstat.pgscan_kswapd_dma[0] + vmstat.pgscan_kswapd_normal[0] + vmstat.pgscan_kswapd_movable[0]"
pgscan_d="vmstat.pgscan_direct_dma[0] + vmstat.pgscan_direct_normal[0] + vmstat.pgscan_direct_movable[0]"
pgscan_k1="vmstat.pgscan_kswapd_dma[1] + vmstat.pgscan_kswapd_normal[1] + vmstat.pgscan_kswapd_movable[1]"
pgscan_d1="vmstat.pgscan_direct_dma[1] + vmstat.pgscan_direct_normal[1] + vmstat.pgscan_direct_movable[1]"
pgscanrate="(pgscan_k + pgscan_d - pgscan_k1 - pgscan_d1) / (cpustat.total_ticks[0] - cpustat.total_ticks[1])"

CMM_MIN="0"
CMM_MAX="1245184"

CMM_INC="meminfo.MemFree  / 40"
CMM_DEC="meminfo.MemTotal / 40"

MEMPLUG = "pgscanrate > 20"
MEMUNPLUG = "meminfo.MemFree > meminfo.MemTotal / 10 "
```

References

This section provides information about where you can find information about topics referenced in this white paper.

- Man pages (SLES11 SP2, RHEL 6.2 or newer distributions):
  - man cpuplugd man
  - cpuplugd.conf
- *Linux on System z: Device Drivers, Features, and Commands*
  `http://public.dhe.ibm.com/software/dw/linux390/docu/l3n1dd13.pdf`

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.

Trademarks

IBM, the IBM logo, ibm.com®, DB2, DB2 Universal Database, DS8000®, ECKD, Express®, FICON®, HiperSockets™, Rational® Redbooks®, Resource Link®, Service Request Manager®, System Storage®, System x®, System z, System z9®, System z10®, Tivoli®, WebSphere, z/VM, and z9® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A complete and current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Centrino, Intel Xeon, Intel SpeedStep, Itanium, Pentium, and Xeon are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may

not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of the manufacturer.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of the manufacturer.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any data, software or other intellectual property contained therein.

The manufacturer reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by the manufacturer, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

THE MANUFACTURER MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THESE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

Index

Printed in USA