



**Dino Quintero
Dave Brelsford
Dean Burdick
John Lewars
Bill Tuel
Curt Vinson
Brian Marcinkowski**

Controlling Application Memory Consumption

Why this paper

pSeries® nodes with large memories are often used for applications that expect to use all the available real memory, either for one serial application or in aggregate for the tasks of a parallel MPI job running on the node.

If an application accesses real memory beyond the amount available, AIX® 5L™ provides support for paging out less frequently used pages. However, paging is relatively slow compared to central processing unit (CPU) speeds, and adversely affects the critical performance requirement of these applications. Excessive paging can disrupt efficient application execution.

This paper describes mechanisms to help applications use maximum available real memory while protecting the integrity of the node. We achieve this by reducing the risk of paging by applications that demand more real memory than is available.

Overview

In this Redpaper we address the following items:

- ▶ We describe how to determine the base amount of real memory available on a node that is operational with all required system daemons and devices initialized, but no application work running. There are some differences between the procedure for AIX 5L Version 5.2 and AIX 5L Version 5.3.
- ▶ We describe the characteristics of the Workload Manager (WLM) component in AIX 5L that relate to real memory control.
- ▶ We discuss the control of parallel MPI jobs using the Parallel Environment for AIX 5L and LoadLeveler® products. There is a subsection on configuring LoadLeveler to use the WLM absolute real memory limit to control memory used by LoadLeveler jobs. There is also a subsection on how to specify memory requirements for LoadLeveler jobs.
- ▶ We discuss related topics and suggestions for future investigations.

Determining available real memory

This section describes the process for determining the available real memory for jobs using small (4 KB) pages on an AIX 5L system. We focus on environments where jobs are run using small pages only. The procedure attempts to size the memory footprint of the system so that any unallocated large pages defined on the system are preserved and paging minimized.

Following we explain the terms pinned and page, which we use in the process for determining available real memory:

- ▶ The use of the term *pinned* refers to a specific flag that prevents a page from being paged-out. Although the **svmon** command reports all large pages as being pinned, an AIX 5L Version 5.3 PTF changed large pages to not always be technically 'pinned'—even though they cannot be paged-out. For the purpose of this procedure, any large pages that the user defines are intended to be left intact. No distinction is made between pinned large pages and *non-pinned* large pages. We use the term *pinned and large pages* to avoid complexity in this paper.
- ▶ For the **svmon -G** command, the first (memory) line of output reports a single 16 MB large page as 4096 4KB units (4096 small pages). For the purposes of this paper, we use the term 4KB units to refer to memory. Also, the term *page* refers to a small (4 KB) page unless otherwise specified.

Steps for determining available real memory

Use the following steps to determine available real memory:

- ▶ Look at the amount of memory that AIX reports.
- ▶ Subtract the pinned or large pages.
- ▶ Subtract the number of pages that WLM reports are assigned to other classes (pages used by the system):

total pages - (non-pageable pages) - (pages used by the system)

1. Make sure the system is idle (no user jobs are running), and execute the **wlmcntrl** command to make sure WLM is enabled. It is important to check that the memory footprint of the system is sized after WLM is enabled because enabling WLM results in real memory allocations that need to be included in the sizing. For example:

```
c89f10sq04:/ # wlmcntrl
```

(no output will be displayed if the command completes successfully)

2. Run the command **svmon -G** to collect data for use in the next two steps. The first row of 'memory' data displayed is all that is required for this procedure.

The **svmon** command is part of the bos.perf.tools fileset, which is installed by default. If this fileset was uninstalled, you must install it again. Depending on what level of AIX 5L is installed, the '**svmon -G**' command output can have two different output formats.

Example 1 and Example 2 on page 4 are examples of both formats:

Example 1 shows an older AIX 5L level format. Example 2 on page 4 shows a more recent AIX 5L level format, although we really only need the first row of data (which is the same for both formats):

- On older AIX 5L levels (any level prior to bos.perf.tools 5.3.0.30, which is part of AIX 5L Version 5.3 ML2), when large pages are defined, there is only one row of output in the last '**svmon -G**' output section that displays the *pgsize* data. If the system has no large pages defined, this last section is not present in the output. Example 1 is from an AIX 5L Version 5.2 system that shows 210 large pages are defined and 153 of these large pages are free (unallocated):

Example 1 AIX 5L Version 5.2 svmon -G output example

```
[c704f2n01][/]> svmon -G
```

	size	inuse	free	pin	virtual
memory	1835008	1081767	753241	951568	1264300
pg space	1703936	1245			
	work	pers	clnt	lpage	
pin	91408	0	0	860160	

in use	170689	19614	31304	233472
	pgsize	size	free	
1page pool	16 MB	210	153	

In Example 1 on page 3, we only need the first row of *memory* data, which gives us the following two values:

- The *size* value (1835008) gives us the total amount of memory available to AIX, expressed in terms of 4 KB units.
 - The *pin* count (951568) gives us the total amount pinned and large-page memory, expressed in terms of 4 KB units.
- In more recent AIX 5L levels (starting with bos.perf.tools 5.3.0.30, which is part of AIX 5L Version 5.3 ML 2), **svmon -G** shows multiple rows of output in the *pgsize* section when there are large pages defined. Again, we only need the first row of memory data but following is an example of the new style of **svmon** output, which we continue to use for calculations in the ensuing steps as shown in Example 2.

Example 2 svmon -G starting with bos.perf.tools 5.3.0.30

```
c89f10sq04:/ # svmon -G
      size      inuse      free      pin      virtual
memory  8093696  1888459  6205237  1605124  1039341
pg space 131072    3015

      work      pers      clnt
pin     789793      0        227
in use  1039341      851     33163

PageSize PoolSize  inuse  pgsp   pin   virtual
s  4 KB   -      839883  3015  556548  805869
L 16 MB   256    57      0     256    5
```

In Example 2, we only need the first line of *memory* output, which gives us the following two values:

- The *size* value (8093696) gives us the total amount of memory available to AIX, expressed in terms of 4 KB units.
 - The *pin* count (1605124) gives us the total amount pinned or large-page memory, expressed in terms of 4 KB units.
3. Determine the amount of memory available by looking at the *size* column of the first row of **svmon -G memory** output from step (2). This *size* is expressed as a number of 4KB units.

In Example 2 on page 4, in the **svmon -G** output, there are 8093696 4KB units of memory available.

4. Determine the number of pages that are pinned or large pages by looking at the *pin* column of the first row of *memory* output from **svmon -G**.

In Example 2 on page 4, in the **svmon -G** output, there are 1605124 4KB units that are large pages or pinned.

5. To get the pageable pages that WLM reports assigned to other classes, and not available to users jobs, subtract the pinned or large page value determined in step (4), from the total memory available determined in step (3) as follows:

(memory available to AIX from step 3) - (pinned or large pages from step 4)

Using the example values from steps (3) and (4), which were obtained from Example 2 on page 4 **svmon -G** sample output, we get the following:

(memory available to AIX from step 3) - (pinned or large pages from step 4)

$(8093696) - (1605124) = 6488572$

6. To determine the number of pages available for jobs, subtract the number of pages assigned to the system from the total pageable pages. The **wlmstat -Smv** command can determine the number of pages assigned to the following classes:

- *System*: pages associated with kernel and root processes
- *Shared*: shared library pages
- *Default*: pages not assigned to any other class are placed in the default class
- *Unclassified*: pages in global segments that currently do not have an owner and are pageable

The *npg* column of the **wlmstat -Smv** output provides the number of pages assigned to a class.

The calculation is: (pageable pages from step 5) - (System pages from **wlmstat -Smv**) - (Shared pages from **wlmstat -Smv**) - (Default pages from **wlmstat -Smv**) - (Unclassified pages from **wlmstat -Smv**) from the output found in Example 3 in the *npg* column.

Example 3 uses the sample from step (5), and the following **wlmstat** output:

Example 3 wlmstat -Smv output

```
wlmstat -Smv
CLASS tr i #pr MEM sha min smx hmx des rap urap npg
Unmanaged 0 0 0 10 -1 1 100 100 99 81 96 790230
```

Default	0	0	2	0	-1	0	100	100	100	100	0	1492
Shared	0	0	0	0	-1	0	100	100	100	100	0	6023
System	0	0	97	4	-1	1	100	100	99	92	40	261302

Using the formula we get the following:

(pageable pages) - (System pages) - (Shared pages) - (Default pages) - (Unclassified pages)

= 6488572 - (261302) - (6023) - (1492) - (0) = 6219755

So there are 6219755 pages (24879020 KB, or 24295 MB) available for user jobs.

Note: There no unclassified pages in the previous calculation.

We use the memory available for user jobs calculated here as documented in “Configuring LoadLeveler to use the AIX 5L Workload Manager absolute real memory limit to enforce memory consumption” on page 9.

7. If the system is going to run without WLM enabled at this point, WLM can be turned off by running `wlmcntrl -o` command, for example:

```
c89f10sq04:/ # wlmcntrl -o
(no output will be displayed if the command completes successfully)
```

Note: The number of shared and system pages may change depending on the workload of the system. It is recommended that you repeat the above steps later to see if the memory available for running jobs changes over time.

For more information about how to configure the memory available for users jobs as LoadLeveler consumable memory, refer to “Configuring LoadLeveler to use the AIX 5L Workload Manager absolute real memory limit to enforce memory consumption” on page 9.

Other considerations

One use of memory that this procedure does not take into account is the use of the AIX file cache to store file pages. In general (GPFS files being an exception) file pages are stored in the AIX file cache. These pages can remain resident in the file cache even after the associated files are closed and the processes accessing the files have exited. Following the application of WLM APARs IY80669 (for AIX 5L Version 5.2), or IY79754 (for AIX 5L Version 5.3), these file pages are not counted by WLM in its enforcement of real memory limits.

An increase in file caching may cause the system to page if all of the available memory is allocated to user jobs. When this occurs (specifically when the minimum number of pages on the free list is reached) AIX's page replacement algorithm is activated, and page replacement occurs. You can lower the **vmo** tunables 'maxclient' and 'maxperm' to favor keeping computational pages over file pages during page replacement.

Note: 'maxclient' and 'maxperm' are not hard limits on these page types unless the corresponding 'strict_maxclient' and 'strict_maxperm' tunables are set to '1', and this is not recommended without AIX performance development's input.

The **vmstat -v** command can be used to sample the 'numperm percentage' and 'numclient percentage' levels on the system during different workloads, to assist in tuning 'maxclient' and 'maxperm'. Refer to the **vmo** main page for more details.

Memory Management with AIX 5L and WLM with Absolute Real Memory Limits

You can use WLM to manage memory by limiting the amount of real memory that each class may consume. The following three types of limits are currently supported:

- ▶ Percentage-Based Memory Limit
- ▶ Absolute Real Memory Limit
- ▶ Large Page Limit

This RedPaper focuses on Absolute Real Memory Limit.

Percentage-Based Memory Limit

This limit is based on class usage of small, pageable pages relative to the number of small pageable pages on the system. This includes all unpinned memory used by the class including stack, data, text, file (persistent and client), and shared memory segments. This limit is enforced in the page stealer (LRU) by selectively stealing pages from classes that are over their limit. Both hard and soft limits are supported. A class may exceed its soft limit as long as there is memory available. A class may not exceed its hard limit, even if there is memory available.

You can specify these limits in the WLM limits file. Use the `wlm_get_info()` API or the **wlmstat** command to see the class usages in percentages or number of pages.

Absolute Real Memory Limit

This limit is based on megabyte (MB) class usage of working storage, including pinned and unpinned, small and large pages. When this limit is reached, the job (class) is killed immediately with no warning or grace period. Notification that a class was killed, including the class name, is provided via RMC through the IBM.WLM resource class. This event, including the class ID, is also recorded in the AIX error log as a WLM_KILL event. This limit can be set using the `wlm_load()` API.

The class usage for real pages can be obtained with the `wlm_get_info()` API or the `1srsrc` command. When a class is killed for reaching this limit, it must be recreated before it can be used. Any process assigned to a class that was killed gets killed also.

Note: Using percentage-based hard limit along with this limit may prevent the job from being killed as expected since the class may get paged if it reaches the percentage limit first.

Large Page Limit

Based on megabyte (MB) usage of large pages only, when this limit is reached the large page allocation fails. Use the `wlm_load()` API to set this limit. The class usage for large pages can be obtained with the `wlm_get_info()` API.

Predefined WLM classes

This section explains the predefined WLM classes.

Unclassified

Contains pages in global segments (for example, files or shared memory) that currently do not have an owner.

Unmanaged

Contains a count of pinned pages, including the 4K pages that were instantiated within large pages, and pages that may not be part of any segment control block (SCB). These pages are excluded when calculating percent usage for all other classes.

Shared

Contains pages of shared library text, and pages from any segment in use by more than one superclass.

System

Contains pages of pageable kernel data and all root processes that are not, by rule, assigned to another class.

Default

Contains pages of all non-root processes that are not, by rule, assigned to another class.

Note:

- ▶ The memory consumption for these classes may increase over time depending on the workload.
- ▶ The WLM page counts are in units of 4K pages.
- ▶ WLM only reports pages that are in use and does not report available memory. Refer to the “Determining available real memory” on page 2 for information about determining how much memory is available.

Running jobs with LoadLeveler and WLM configured

You can configure LoadLeveler to use AIX 5L Workload Manager (WLM) to give greater control over CPU and real memory resource allocation. In this paper we concentrate on how you can use LoadLeveler and WLM to exercise control over real memory resource allocations.

When configured to use WLM, LoadLeveler uses the WLM API to start WLM and to dynamically generate WLM classes. A single WLM class is created for each job step. This is done for each machine that is assigned to execute a job step. LoadLeveler then defines limits for that class depending on the defined LoadLeveler enforcement policy and the consumable memory requested by the job.

To use LoadLeveler in this way, the administrator must configure real memory consumable resources, and specify that the resources need to be enforced.

Configuring LoadLeveler to use the AIX 5L Workload Manager absolute real memory limit to enforce memory consumption

Use the following steps to configure LoadLeveler to use AIX 5L WLM and the WLM absolute real memory limit to enforce memory consumption. By using this method, an installation can reduce paging during the execution of user applications.

1. Define ConsumableMemory as a consumable resource in the *SCHEDULE_BY_RESOURCES* global configuration keyword. This enables the LoadLeveler scheduler to consider consumable resources. There are four consumable resource types that may be specified with this keyword, ConsumableCpus, ConsumableMemory, ConsumableVirtualMemory and RDMA. Any combination of these resource types may be specified. In order for the LoadLeveler scheduler to consider real memory as a consumable resource, ConsumableMemory must be specified in this keyword.
2. Define ConsumableMemory in the *ENFORCE_RESOURCE_USAGE* global configuration keyword. This enables enforcement of consumable resources by AIX 5L WLM. There are two consumable resource types that may be specified with this keyword, ConsumableCpus, and ConsumableMemory. In order for LoadLeveler to use AIX 5L WLM to enforce consumption of real memory, ConsumableMemory must be specified in this keyword.
3. Set the *ENFORCE_RESOURCE_MEMORY* configuration keyword to true. This setting allows AIX 5L WLM to limit the real memory usage of a WLM class by setting the WLM absolute real memory limit. When a class exceeds its limit, all processes in the class are killed.

Note: You must define ConsumableMemory in the *ENFORCE_RESOURCE_USAGE* keyword in the global configuration file, or LoadLeveler does not consider the *ENFORCE_RESOURCE_MEMORY* keyword to be valid.

4. Set the *ENFORCE_RESOURCE_SUBMISSION* configuration keyword to true. This setting forces all LoadLeveler jobs to specify requirements for consumable resources that are being enforced (in this case, ConsumableMemory).
5. In order for LoadLeveler to schedule by resources and to enforce resource consumption, you must specify the consumable resources available on a machine by coding the *resources* keyword in the machine stanza in the LoadLeveler administration file as shown in Example 4. To specify the amount of consumable real memory available on a machine, specify the ConsumableMemory reserved word followed by a count representing the amount of real memory available. The amount specified for ConsumableMemory should be the amount determined by the method outlined for estimating memory available for application use. Refer to the "Determining available real memory" on page 2".

Example 4 Consumable resources keyword

```
example: resources = ConsumableMemory(13 GB)
```

You can specify ConsumableMemory with both a count and units. The count must be an integer greater than zero. The allowable units are those normally used with LoadLeveler data limits:

- b bytes
- w words
- kb kilobytes (2^{10} bytes)
- kw kilowords (2^{12} bytes)
- mb megabytes (2^{20} bytes)
- mw megawords (2^{22} bytes)
- gb gigabytes (2^{30} bytes)
- gw gigawords (2^{32} bytes)
- tb terabytes (2^{40} bytes)
- tw terawords (2^{42} bytes)
- pb petabytes (2^{50} bytes)
- pw petawords (2^{52} bytes)
- eb exabytes (2^{60} bytes)
- ew exawords (2^{62} bytes)

Note: If no resources are defined for a machine, then no enforcement is done on that machine.

If the count specified by the administrator is greater than the total memory LoadLeveler determines is available on a machine, the count value is reduced to match the total memory.

Summary of steps to configure LoadLeveler

Specify the following statements in the LoadLeveler configuration file:

```
ENFORCE_RESOURCE_USAGE = ConsumableMemory
ENFORCE_RESOURCE_MEMORY = true
SCHEDULE_BY_RESOURCES = ConsumableMemory
ENFORCE_RESOURCE_SUBMISSION = true
```

Specify the following statement in machine stanzas of the LoadLeveler administration file:

```
resources = ConsumableMemory(13 GB)
```

Note: 13 GB is an example, each installation needs to define the memory available on its machines.

Specifying consumable memory requirements in LoadLeveler

Following are three ways to specify consumable memory for LoadLeveler jobs:

1. In the LoadLeveler job command file, specify the amount of consumable memory required per task, by specifying a count for ConsumableMemory in the #@resources JCF statement. The count must be an integer greater than zero. For example:

Job command file:

```
#@ resources = ConsumableMemory(2 GB)
```

2. Specify the class name in the #@class JCF statement of the job command file of a class defined in the LoadLeveler administration file. This is where a per task ConsumableMemory is defined, in the default_resources statement in the corresponding class stanza. For example:

LoadLeveler administration file:

```
GB2_Class: type = class
  default_resources = ConsumableMemory(2 GB)
```

Job command file:

```
#@ class = GB2_Class
```

3. If invoking POE interactively, specify the class name in the LOADL_INTERACTIVE_CLASS environment variable of a class defined in the LoadLeveler administration file. This is where a per task ConsumableMemory is defined in the default_resources statement in the corresponding class stanza. For example:

LoadLeveler administration file:

```
GB2_Class: type = class
  default_resources = ConsumableMemory(2 GB)
```

Environment variable:

```
LOADL_INTERACTIVE_CLASS = GB2_Class
```

You can always specify ConsumableMemory with both a count and units. The count must be an integer greater than or equal to zero.

Consumable memory requirements are always specified per task; however, memory consumption is enforced for a single WLM class defined for the job step. This means that if you have an unbalanced application where each task consumes different amounts of memory, you can still specify a per task consumable memory requirement. That memory requirement is multiplied by the number of tasks to determine the total memory requirement for the job step. For example:

One task may need 6 GB and remaining three tasks need only 2 GB (total requirement is 12 GB). Specifying ConsumableMemory (3 GB), generates a WLM class with a memory limit of 12 GB, for a job step with four tasks running on a machine, which satisfies the unbalanced requirement.

Determining when a LoadLeveler job is killed for exceeding the WLM absolute real memory limit

When LoadLeveler is configured to use the WLM absolute real memory limit to control real memory consumption by LoadLeveler job steps, the operating system can kill LoadLeveler job steps that exceed the absolute real memory limit with a SIGKILL signal.

Ordinarily the owner of the job is unable to tell that the job step was killed for this reason.

A LoadLeveler administrator can write prolog and epilog installation exits that can run before and after LoadLeveler job steps run, respectively. In this case we are interested in the epilog user exit, since we want to take action depending on how the job step terminates. The administrator can write a LoadLeveler epilog exit to detect when WLM kills a LoadLeveler job step and to take action when it detects this, such as send mail to the owner of the job step.

To configure an epilog program, specify `JOB_EPILOG = epilog_program` in the LoadLeveler configuration file. Where `epilog_program` is the name of the epilog program that is invoked after a job step terminates.

In this case the epilog program must determine if the job step was killed because it exceeded the WLM absolute real memory limit. A simple C program, which uses the `wlm_get_info()` API interface, can determine that. We provide a sample of such a program in Example 5 on page 14.

There are several LoadLeveler environment variables that are set when the LoadLeveler epilog exit is invoked. Three of them are of interest to the following sample epilog shell script:

LOADL_PID - LoadLeveler creates a WLM class for every job step, using the process id of the LoadLeveler starter process to name the WLM class. The LOADL_PID environment variable contains the LoadLeveler starter process process id.
LOADL_STEP_ID - Is the LoadLeveler step identifier.
LOADL_STEP_OWNER - The username of the owner of the job step.

Example 5 is a shell script that can serve as a LoadLeveler epillog to notify a user when WLM kills a LoadLeveler job.

Example 5 Sample user notification shell script

```
#!/bin/ksh

# Run the wlm_epilog program to determine if the user's application
# was killed by wlm for exceeding its real memory limit.
# The wlm_epilog program is a simple program which uses the
# wlm_get_info() api interface and returns 1 if the specified wlm class was
# killed by WLM.
outstring=`/var/loadl/bin/wlm_epilog $LOADL_PID`

if [ "$?" -eq 1 ]
then

# Need to export the sendwait environment variable to prevent the mail
# command from forking a background process. Any background process
# started by the epillog may be terminated if the LoadLeveler process
# tracking feature is enabled.
export sendwait
mail -s "Job $LOADL_STEP_ID killed" $LOADL_STEP_OWNER <<eod
LoadLeveler job step $LOADL_STEP_ID was assigned to WLM class $LOADL_PID.

$outstring.

LoadLeveler job step $LOADL_STEP_ID was killed by the operating system
for exceeding its real memory limit.
eod
fi
```

Important: Example 5 demonstrates how an administrator can write a LoadLeveler epillog to notify users when WLM kills LoadLeveler jobs. The sample as is may not be suitable for an installation. A drawback to this particular sample is that mail is sent to the user from EVERY node where WLM kills the user processes.

Example 5 on page 14 epilog shell script uses an installation written binary "/var/loadl/bin/wlm_epilog" that returns 1 if the WLM class was killed or 0 if it was not.

The following is a sample of a simple C program that uses the wlm_get_info() API interface to determine if the WLM class was killed. This sample program takes the name of the WLM class to check as its only argument.

Example 6 C program sample

```
/*
 * Sample program to determine if a WLM class has been killed
 */

#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <malloc.h>
#include <sys/wlm.h>

/*
 * This program expects one argument, the wlm class name to query
 *
 * The WLM wlm_get_info api interface is used to return wlm information
 * for the wlm class passed as an argument.
 *
 * If the wlm_info.i_cl_state is equal to CL_SKILLED, then the WLM class
 * was killed by WLM. The program exits with exit code 1 in this case.
 *
 * Exit code:
 *      0 - WLM class has not been killed
 *      1 - WLM class HAS been killed
 *     -1 - call to WLM api failed
 */
int main(int argc, char **argv, char **envp) {
    int wlm_flags = WLM_VERSION;
    struct wlm_args wlm_input;
    struct wlm_info wlm_output;
    int count = 1;
    int rc = 0;
    int exit_code = 0;
    int memlimit = 0;
    char *memunits = NULL;

    memset(&wlm_input, 0, sizeof(wlm_input));
    memset(&wlm_output, 0, sizeof(wlm_output));

    /*
     * Initialize wlm

```

```

    */
    rc = wlm_initialize(wlm_flags);
    if (rc != 0) {
        fprintf(stderr, "wlm_initialize failed rc= %d.\n", rc);
        exit(-1);
    }
    rc = 0;

    /*
    * Get wlm information for the wlm class
    */
    wlm_input.versflags = WLM_VERSION | WLM_SUPER_ONLY;
    strcpy(wlm_input.cl_def.data.descr.name, (char*)argv[1]);

    rc = wlm_get_info(&wlm_input, &wlm_output, &count);
    if (rc != 0) {
        fprintf(stderr, "wlm_get_info failed for wlm_class %s. errno =
%d, rc = %d\n",
                wlm_input.cl_def.data.descr.name, errno, rc);
        exit_code = -1;
    } else {
        if (count == 0) {
            fprintf(stderr, "wlm_get_info returned a count of 0 for
wlm_class %s.\n",
                    wlm_input.cl_def.data.descr.name);
            exit_code = -1;
        } else if (count == 1) {
            if (wlm_output.i_cl_state == CL_SKILLED) {
                /*
                * WLM class was killed by WLM
                */

                memlimit =
wlm_output.i_descr.res[WLM_RES_TOTALRMEM].hardmax;
                memunits =
wlm_output.i_descr.res[WLM_RES_TOTALRMEM].wlm_u.unit;

                /*
                * At the time this sample program was written, there was a bug
                * in the wlm_get_info api interface in that it returns the hardmax
                * value in pages, not the units specified in wlm_u.unit. So the following
                * message would not be clear. An alternate fprintf statement is provided
                * as a substitute until wlm_get_info is changed.

                fprintf(stdout, "WLM class, %s, killed because memory consumed exceeded the WLM
absolute memory limit (%d %s)\n",
                        wlm_input.cl_def.data.descr.name,
                        memlimit, memunits);
            }
        }
    }

```

```

        */
        fprintf(stdout, "WLM class, %s, was killed
because memory consumed exceeded the WLM absolute memory limit (%d pages)\n",
        wlm_input.cl_def.data.descr.name,
memlimit);
        exit_code = 1;
    }
    } else {
        fprintf(stderr, "wlm_get_info returned more than one
WLM class.\n");
        exit_code = -1;
    }
}

exit(exit_code);
}

Makefile to build wlm_epilog.

CC = cc
CFLAGS = -g
LDFLAGS = -lwlm

wlm_epilog: wlm_epilog.o
    $(CC) -o $@ $? $(LDFLAGS)

```

Resource Monitoring on AIX 5L

The Resource Monitoring and Control (RMC) subsystem provides a framework that makes it easy to monitor and respond to resource related events on your system. With RMC, resources are abstracted into resource classes, each of which may contain one or more resources that can be monitored. Each resource is represented as an instance of a resource class and has a set of attributes associated with it. These attributes can be used in expressions to define conditions of interest for which notification can be sent.

The RMC command line interface provides all of the commands necessary for monitoring and configuring responses to events for a large number of resource classes. The `lsrsrc` command can be run to see the resource classes that are supported on your system.

RMC is supported in both standalone and clustered environments on AIX 5L Version 5.1 and later. Refer to "Managing and monitoring resources using RMC

and resource managers" in chapter 4 of the *IBM Reliable Scalable Clustering Technology (RSCT) Administration Guide*, SA22-7889-09 for more information about resource monitoring.

Monitoring WLM with RMC

When WLM is running, each WLM class is exported to RMC as an instance of the IBM.WLM resource class. The dynamic attributes of each instance (WLM class) can be used in the definition of conditions to provide notification that a WLM class reached a certain threshold.

To see the dynamic resource attributes refer to Example 7.

Example 7 Attributes that can be monitored for the IBM.WLM resource class

```
# lsrsrc -Ad IBM.WLM
resource 3:
    CPUPct           = 0
    CPUAtSoftMax    = 0
    CPUAtHardMax    = 0
    MemPct          = 13
    MemAtSoftMax    = 0
    MemAtHardMax    = 0
    DiskIOPct       = 0
    DiskIOAtSoftMax = 0
    DiskIOAtHardMax = 0
    NumProcs        = 58
    NumProcsAtMax   = 0
    NumThreads      = 150
    NumThreadsAtMax = 0
    NumLogins       = 30
    NumLoginsAtMax  = 0
    RealMemInUse    = 38703
    RealMemAtMax    = 0
    VirtMemInUse    = 38703
    VirtMemAtMax    = 0
    ProcAtMax       = 0
```

Notification of the Real Memory Limit Event

The 'RealMemAtMax' attribute is a boolean that is asserted whenever a class is killed because it reached its real memory limit. This attribute can be monitored for a value of "1" to receive notification that this event happened.

So for example, to define a condition that is true when a class gets killed, you might do the following:

1. Define the condition.

```
# mkcondition -r IBM.WLM -e "RealMemAtMax = 1" -E "RealMemAtMax = 0" "RML Reached"
```

2. Define a response.

```
# mkresponse -n "Call Script" -s "/usr/bin/<my_script>" -ea "RML Response"
```

3. Associate the condition and the response.

```
# mkcondresp "RML Reached" "RML Response"
```

4. Start monitoring.

```
# startcondresp "RML Reached"
```

After monitoring begins, the response script `<my_script >` is called when any WLM class reaches its RML (been killed). In the execution context of the response script, a number of environment variables starting with the name *ERRM_* are set that provide information about the event. In particular, *ERRM_RSRC_NAME* contains the name of the WLM class that was killed, and *ERRM_NODE_NAME* contains the name of the node on which the event occurred.

By default, the monitoring occurs on the local system. The location and scope of monitoring is controlled with the *CT_CONTACT* and *CT_MANAGEMENT_SCOPE* environment variables.

See the main pages for the **mkcondition**, **mkresponse**, **mkcondresp**, and **startcondresp** commands, or refer to "Managing and monitoring resources using RMC and resource managers" in chapter 4 of the *IBM Reliable Scalable Clustering Technology (RSCT) Administration Guide, SA22-7889-09* for more information.

Performance implications

HPC applications try to minimize non-application use of resources, so it is natural to wonder whether activating WLM to control real memory usage has significant impact on application performance.

WLM real memory control is invoked when the application accesses a memory page for the first time. As application memory is assigned real page frames, WLM keeps track of the number of real page frames and terminates the process if the number of real page frames exceeds the assigned limit for the class. So, generally, applications that do not touch memory for the first time during the

critical time sections are not noticeably affected. Applications that dynamically allocate memory during critical time sections may have a minor impact.

To illustrate, we conducted the NAS CG and HPL Linpack benchmark tests.

NAS CG

We ran the NAS CG benchmark on 8 p575 nodes under Parallel Environment MPI with eight tasks per node (total of 64 tasks). The nodes were running AIX 5L Version 5.3, booted in SMT mode, and `MP_TASK_AFFINITY=MCM` was set. We conducted ten runs with each environment.

With WLM disabled, reported execution time ranged from 8.09 to 8.25 seconds, with an average of 8.17 seconds.

With WLM enabled and used with LoadLeveler, reported execution time ranged from 8.15 to 8.26 seconds, with an average of 8.20 seconds.

HPL Linpack

The HPL Linpack benchmark was run on a single p575 node under Parallel Environment MPI with 2 tasks, large pages, SMT off, and a problem size of 48000 with a block of 224. Each run took about 22 minutes. We conducted four runs with each environment.

With WLM disabled, reported performance ranged from 56.20 to 56.21 GFlop, with an average of 56.20 GFlop.

With WLM enabled and used with LoadLeveler, reported performance ranged from 56.15 to 56.19 GFlop, with an average of 56.17 GFlop.

Memory allocation

To bound the effect of WLM on application performance, we wrote and timed a program that merely allocates, touches, and frees memory. The timed loop consists of the following items:

- ▶ Malloc of 100 MB (25600 small pages)
- ▶ Touch the first byte of each page malloced
- ▶ Free the malloced memory
- ▶ [option] "disclaim" malloced memory (`MALLOCOPTIONS=disclaim`)

For a loop count of 100000 loops, having WLM enabled added 0.45% to the measured time (averaging 209 seconds) when the "disclaim" option is not

specified. When the "disclaim" option is specified, having WLM enabled added 10% to the measured time (averaging 190 seconds for 1000 loops). The team knows of no HPC application that uses "disclaim"—the number given here is intended to merely bound the potential impact of WLM.

The overall conclusion is that WLM does not appreciably add to the application time for HPC applications.

The team also verified that the LoadLeveler/Parallel Environment Checkpoint/Restart and Preemption functionality is not affected by having WLM enabled.

APARs required for AIX 5L Version 5.2 and Version 5.3

The following APARs are required when using LoadLeveler, Workload Manager (WLM), and AIX 5L Versions 5.2 and 5.3:

- ▶ AIX 5L Version 5.2
 - IY80669
- ▶ LoadL 3.3.1.1. - contains all relevant fixes
- ▶ AIX 5L Version 5.3 ML4 or AIX 5L APARs that affect WLM
 - AIX 5L Version 5.3
 - IY79754

What the team will address next after the release of this Redpaper

The IBM team is currently looking at approaches to better assess the impact of the file cache in accurately determining the available real memory for jobs when WLM is used to enforce real memory limits.

The team will investigate a process for controlling application memory consumption for jobs that use large pages.

The team that wrote this Redpaper

This Redpaper was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Austin Center.

Dino Quintero is a Senior Certified Consulting IT Specialist at ITSO in Poughkeepsie, New York. Before joining ITSO, he worked as a Performance Analyst for the Systems and Technology Group and as a Disaster Recovery Architect for IBM Global Services. His areas of expertise include disaster recovery and pSeries clustering solutions. He is certified on pSeries system administration and pSeries clustering technologies. He is also an IBM Senior Certified Professional on pSeries technologies. Currently, he leads teams delivering IBM RedBook solutions on pSeries clustering technologies and delivering technical workshops worldwide.

Dave Brelsford is an Advisory Software Engineer with IBM in Poughkeepsie and is a member of the LoadLeveler development team.

Dean Burdick is a member of the AIX kernel development group in Austin, Texas. He is the team lead for development of the process management subsystem and WLM. He has worked for IBM as a Software Engineer for the past nine years, and spent the last seven working in the areas of process and resource management.

John Lewars is a Senior Software Engineer with IBM in Poughkeepsie, and is a member of the Switch Management and LAPI Protocol development teams. Prior to working in HPC development, John has a background on the level-3 service team, developing code fixes and working on client problems in a variety of areas, including performance.

Bill Tuel is the Cluster Performance Architect for the IBM Advanced Cluster Technology Team (ACTT) in Poughkeepsie, and leads the team responsible for performance analysis and reporting of high performance computing systems involving the pSeries High Performance Switch. Previously, he was a team lead for the Parallel Environment product, and was actively involved in High Performance Computing since 1993. Dr. Tuel was a member of the IBM San Jose Research Staff from 1965 to 1976, and a member of the IBM Palo Alto Scientific Center from 1976 to 1992, when he transferred to Product Development.

Curt Vinson is an Executive Program Manager in the HPC System Architecture and Delivery team in Poughkeepsie New York. He has 27 years of experience in IBM mainframe and high performance computer development including 19 years in hardware design team management. He has been a member of the IBM HPC team since 1992. His most recent assignments include Team Manager for the HPS adapter and switch development, followed by program management responsibility for the test and release of the HPS communication fabric. Most recently, he carried the responsibility of HPC Delivery Project Management working with IBM domestic and international account team members and customers to facilitate the successful install and acceptance of new HPC clusters.

Brian Marcinkowski is the High Performance Computing Federation lead for the pSeries Customer Satisfaction Project Office.

Thanks to the following people for their contributions to this project:

Dave Hepkin
IBM Austin

Charles Johnson
IBM Poughkeepsie

Will Weir
IBM United Kingdom

Oliver Treiber
ECMWF, United Kingdom

Nicolas Christin
Groupe Bull, Austin

Kurt Carlson
ARSC, Alaska

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 give 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.

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.

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.

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 illustrates 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. 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 IBM's application programming interfaces.

- Send us your comments in one of the following ways:
- ▶ Use the online **Contact us** review redbook form found at:
ibm.com/redbooks
 - ▶ Send your comments in an email to:
redbook@us.ibm.com
 - ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 905, 11501 Burnet Road
Austin, Texas 78758-3493 U.S.A.



Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX 5L™

AIX®

@server®


@server®

IBM®

LoadLeveler®

pSeries®

Redbooks™

Redbooks (logo) ™

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