Configuring Active Memory Sharing from a customer's experience

IBM's Early Ship Program for Active Memory Sharing on POWER6

Chris Gibson August 04, 2009

Share the experience of being part of the IBM® Early Ship Program for Active Memory Sharing on POWER6™. See how AMS was configured and deployed in a customer's non-production AIX® lab environment.

Introduction

I was fortunate enough to take part in IBM's Early Ship Program (ESP) for Active Memory Sharing (AMS) on POWER6. This article describes how I configured Active Memory Sharing in my non-production AIX lab environment. I'll also touch on performance considerations with AMS. My hope is that as people start to adopt AMS, this article will help to get them started on their journey into this new PowerVM™ virtualization technology.

The organization that I work for was nominated and accepted into the ESP for AMS. For several months we were able to test AMS before anyone else (outside of IBM) had a chance to get their hands on it. Now that AMS is available to AIX and POWER6 customers, I thought it would be worth sharing my experience with the AIX community.

Being involved in the beta program was a great experience. We were given access to the beta code and documentation for testing. We were also able to open PMRs for any bugs we uncovered during the test. A forum was created specifically for ESP customers so that we could post questions and get answers directly from the actual AMS developers.

We were expected to test AMS on our non-production systems and report back on a regular basis, providing feedback on performance, functionality, and usability to the developers.

Overview

AMS is an enhancement to IBM's PowerVM virtualization technology available on the POWER6 platform. It is a feature that some have been eagerly awaiting for quite some time. It is the final piece in the puzzle for a fully virtualized AIX and POWER environment.

AMS intelligently flows memory from one logical partition (LPAR) to another for increased utilization and flexibility of memory. The concept is very similar to that of the shared processor pool and micro-partitions. It allows you to oversubscribe memory on a POWER6 system and let the system (the POWER Hypervisor™) allocate memory where it is needed. No more DLPAR operations are required!

Of course, you need to understand how AMS works and how it interacts with the virtual memory and paging on AIX. Fortunately, existing performance tools have been enhanced to assist in monitoring and managing AMS systems.

So why do we need AMS? Well, consider an environment that has a p6 570 with 256GB of memory and 32 AIX LPARs deployed where each LPAR has 8GB of memory assigned to it. All of the memory has been allocated and cannot be shared with other LPARS. However, we have unallocated processing units that we could use to build more LPARs but we are unable to, as all of the memory has been consumed. The usual course of action here might be to activate more memory (using CUoD) or to buy and install more physical memory.

But wouldn't it be nice to share any "unused" or "idle" memory on an LPAR with other LPARs and give it back when it is really needed? But how do I tell if the LPARs really need all that memory? This is often a hard task, due to AIX optimization of memory. You perform some workload analysis and find that not all of the LPARs are used at the same time. This is a non-production system running a bunch of development and test environments for SAP/Oracle (for example). You now know that although the LPARs may have consumed all of their memory, they are not really using it all of the time.

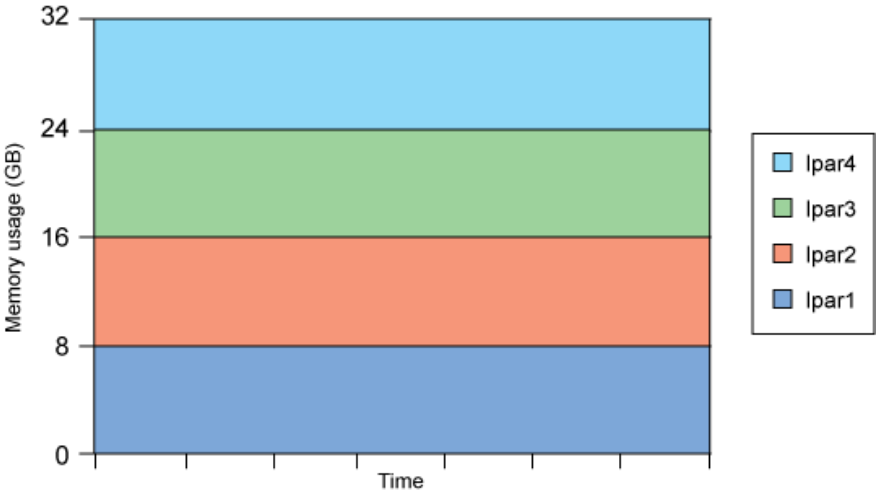
Can we redistribute the "idle" memory and deploy it somewhere else where it is really needed? Yes: With AMS you can.

Active Memory Sharing (AMS)

In this section, I'll provide a very brief overview of how AMS functions. However, I encourage you to review the official IBM documentation relating to AMS for detailed information.

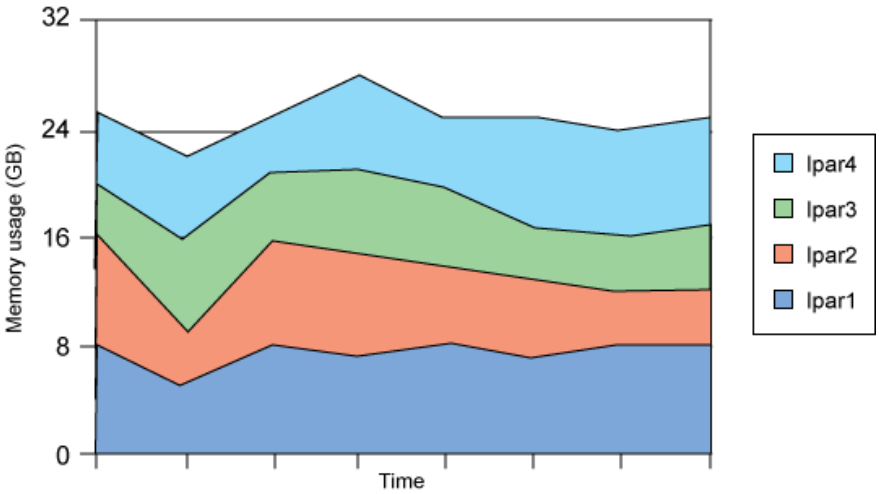
Traditionally, each LPAR owns its memory. Any unused memory is potentially wasted. If memory is overused (over-committed), then AIX will page to paging space (disk). To handle these situations, the AIX administrator could either remove some memory (if he could determine actual usage of memory) and reallocate it to another LPAR using DLPAR or if there was free (unallocated) memory, add it to the LPAR, again using DLPAR. Refer to Figure 1.

Figure 1. LPARs with dedicated physical memory



With AMS, the Hypervisor can take control of memory allocation automatically. The system's physical memory is placed in a "shared memory pool". The LPARs are assigned "logical" shared memory. However, they believe the memory is real, so the change is transparent to the system. This memory can be assigned to LPARs on demand. Unused memory can be used to build more LPARs or allocated to those who need it. The LPARs and the Hypervisor work collaboratively to determine when and where memory should shared. Refer to Figure 2.

Figure 2. LPARs with shared "logical" memory



For AMS to function, a new device called a Paging Virtual I/O Server (VIOS) is required. This Paging VIOS device provides paging services for the shared memory pool and manages the Hypervisor paging spaces for shared memory LPARs. As memory is dynamically managed among multiple LPARs, the Hypervisor has to use a paging device to back up the excess memory that it cannot back up using physical memory from the shared memory pool. This brings us to memory subscription.

There are three ways to configure memory subscription with AMS.

The first is called ***Non over-commit***. In this situation the amount of real memory in the shared pool is enough to cover the total amount of logical memory configured (for instance, four LPARs need 8GB each, so the pool is configured with 32GB of memory. This fits snuggly into the shared memory pool.).

The second method is ***Logical over-commit***. The logical memory "in use" at a given time is equal to the physical memory in the pool. So, the total logical configured memory can be greater than the physical memory in the pool. However, the *working set* never exceeds the physical memory in the pool. More on *working sets* in a moment. In this configuration, the memory actively in use (the working set) on an LPAR will reside in physical memory, with the remainder of the LPARs' logical memory residing on the paging device on the paging VIOS. Just to reiterate the difference between "in use" and "configured" with this method: The "configured" logical memory can be greater but the "in use" memory, at any point in time, does not exceed the memory pool size.

The final configuration is ***Physical over-commit***. In this scenario, the *working set* memory requirements of all LPARs can exceed the physical memory in the pool. Therefore, logical memory has to be backed by both the physical memory in the pool and by the paging devices on the paging VIOS. In the case of "over-commitment," the Hypervisor backs the excess logical memory using the paging devices on the VIOS.

So which method should you use? How do I determine which LPARs are good candidates for AMS? It depends on your workload requirements. Essentially, any workload that is not maximizing its physical memory consumption is an excellent candidate for AMS.

My preference would be to follow the ***Logical over-commit*** approach. This approach works best for workloads that peak at different times, have low average memory residency (*working set*) requirements, and do not have sustained workloads. Typically, non-production development and test environments fit this description. Also, failover or "backup" LPARs (for PowerHA Clusters, for example), that are used for redundancy and only require resources in a failover/takeover situation are also good targets for AMS.

To determine which approach is best you need to have an idea of the working set requirements of your systems. The working set is the memory actually used or required for the workload on your system. Prior to migrating a dedicated LPAR to AMS, you can use AIX performance tools, like svmon, to determine the memory in use on a dedicated memory LPAR.

The ***Physical over commit*** method is OK for workloads that use a lot of AIX file cache, are less sensitive to I/O latency (such as file servers, print servers or network applications) and are inactive most of the time. Perhaps your NIM server is also a good candidate, as it may be used infrequently, only to install AIX systems and perform maintenance?

Based on my tests, I recommend continuing to use dedicated memory for production systems that have the following characteristics: high quality of service requirements, have sustained memory consumption, expect predictable performance and have sustained high CPU utilization and high memory bandwidth requirements. So no, I won't be deploying AMS into my production

environment. Other environments may not be suited to AMS, for example Stress and Volume Testing systems.

Will my workload fit?

In my lab environment, I had to determine if my workload would fit into my shared memory pool. I had two existing LPARs using dedicated memory. Before converting them to shared memory LPARS, I did some numbers. Each LPAR had 4GB of dedicated memory. LPAR1 was relatively idle most of the time and, based on its working set, did not require all 4GB of memory. LPAR2 (also with 4GB of memory) was busier and at times required more memory and occasionally it would page to paging space. It could benefit from more memory.

So in my AMS environment, I decided that I'd add a little extra logical memory to each LPAR, to cater for peaks in demand. LPAR1 would receive 6GB of shared memory and LPAR2 would be allocated 8GB. The shared memory pool would be configured with 12GB of physical memory, with a total of 14GB of logical memory allocated to the LPARs. The total logical memory was larger than the shared memory pool. But given the workload of each LPAR, it should be fine in most situations.

In most cases both LPARs will fit happily together in the memory pool. If the working set of both LPARs remains less than or equal to the pool size, then the there is no over-commit. If the working sets become slightly larger than the pool size, then some paging will occur as the Hypervisor rebalances the memory usage across the LPARs. If the working set of the LPARs becomes much larger than the size of the pool, then there will be lots of paging and performance could suffer greatly. Hence, it is important to understand your workload well before moving to AMS.

AMS works with the AIX virtual memory manager and the Hypervisor to manage memory allocation. If the workload of all the LPARs is slightly larger than the pool, then the Hypervisor will ask AIX for help in determining where pages can be freed. AIX can loan pages to the Hypervisor as required. LPARs that have a higher demand for memory will be favored by the Hypervisor when distributing the memory.

The Hypervisor will aggressively steal pages from LPARs if the pool is physically over-committed. If this impacts performance dramatically, it's probably time to add more physical memory to the pool.

Preparation and planning

If you plan to use AMS in your environment, you must ensure that you have the following required hardware, AIX version, VIOS version, firmware level, and virtualization configuration for your systems:

- An IBM Power System based on the POWER6 processor
- Enterprise PowerVM activation for Active Memory Sharing
- Firmware level 340_070
- HMC version 7.3.4 for HMC managed systems
- Virtual I/O Server Version 2.1.0.1-FP20.0

- AIX 6.1 TL3
- Micro-partition only
- Virtual I/O only

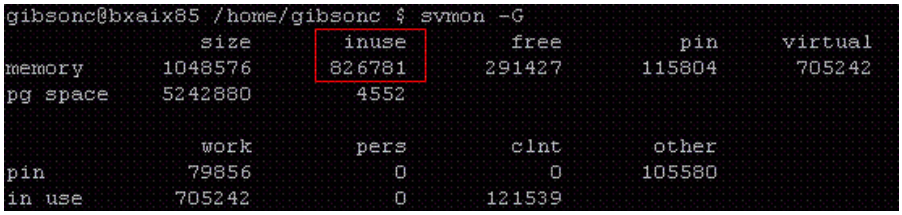
My non-production lab environment consisted of a JS22™ blade with 16GB of memory and four POWER6 processors. Configured on the blade were a VIOS (IVM) and two AIX 6.1 LPARs. As part of the ESP, I was given beta code to install, which would enable AMS on my JS22, VIOS, and AIX LPARs. I updated the lab to the supplied beta levels:

- Upgraded the blades firmware to EA340_043_039.
- Upgraded VIOS on blades to 2.1.0.1-FP-20.0.
- Applied AMS efixes to VIOS and AIX LPARs.
- Applied VET code for AMS activation.

Both LPARs were existing systems (using dedicated memory), with existing workload. The first LPAR, bxaix85, was running an SAP/Oracle instance. The other LPAR, bxaix86, was running three applications: SAP/Oracle, Wily, and Oracle Grid Control, where each application was housed in a Workload Partition (WPAR).

The working set for both LPARs came to roughly 9.3GB, which would fit nicely into the pool. I ran `svmon -G` and observed the in-use memory value to determine the working set. Refer to Figure 3.

Figure 3. svmon output displaying dedicated memory in use on an LPAR



There were rare occasions where the LPARs working set would grow and be slightly larger than the size of the pool. This would be a good test of AMS and how it performs. My aim was to migrate these existing LPARs from dedicated to shared memory and monitor their performance.

Configuring Active Memory Sharing

To configure AMS in my lab, I executed the following steps:

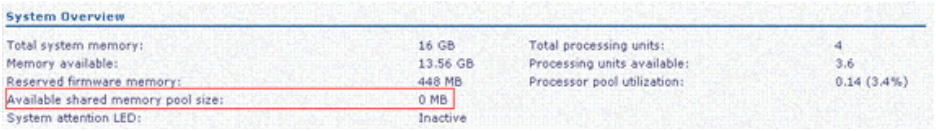
- Verified that the AMS VET code was activated successfully using the lsvet command on the VIOS. Refer to Figure 4.

  ### Figure 4. Verifying AMS is enabled

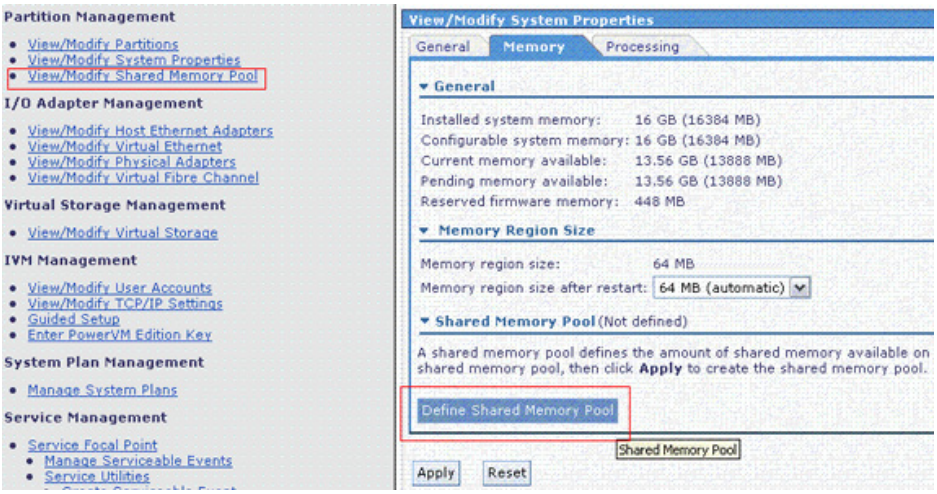
- Defined a shared memory pool. The entire AMS configuration was performed using the Integrated Virtualization Manager (IVM) interface.
  I decided on a shared memory pool size of 12GB. Prior to configuring the pool, the available pool size shows 0MB. Refer to Figure 5.

Figure 5. Shared memory pool size of zero prior to pool creation



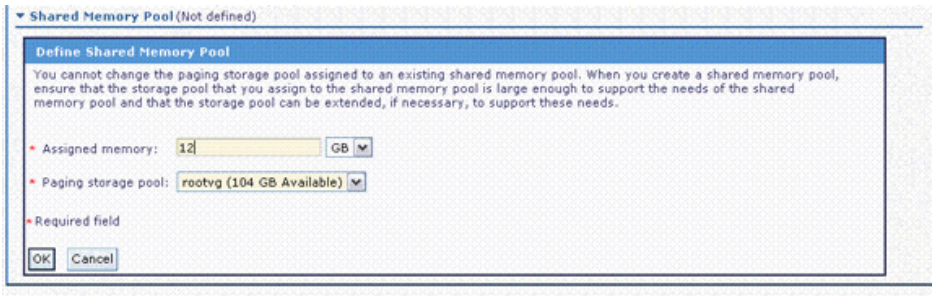
Defining the pool was straightforward with the IVM interface. Refer to Figure 6.

Figure 6. IVM interface for shared memory creation



I specified **12GB** for the pool and chose **rootvg** as the location for the VIOS paging devices. I recommend that you use high-performance SAN disk for this purpose, but for my beta test, this would do just fine. Refer to Figure 7.

Figure 7. Defining the shared memory pool



With my pool defined, I observed its settings from the IVM and the VIOS. Notice that I also changed the maximum size to **16GB** so that I could grow the size of shared memory pool dynamically if need be. Refer to Figures 8 and 9.

Figure 8. Shared memory pool settings

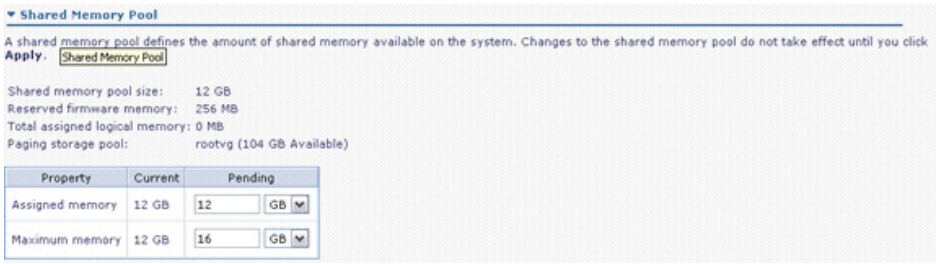


Figure 9. Shared memory and paging pool view from the VIOS

```
$ lshwres -r mempool -F curr_pool_mem,paging_storage_pool
12288,rootvg
```

- The next step was to migrate my existing dedicated-memory LPARs to shared-memory LPARs. To do this, I would first need to shut down each LPAR and change the profile from dedicated to **shared**. Refer to Figures 10, 11, and 12.

Figure 10. LPAR must be inactive before changing its profile

Figure 11. Select Shared in the LPARs' memory profile

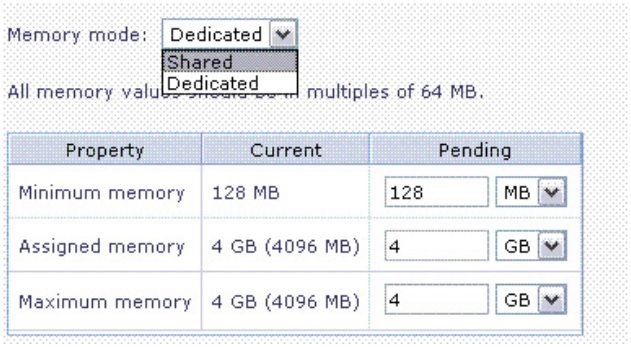
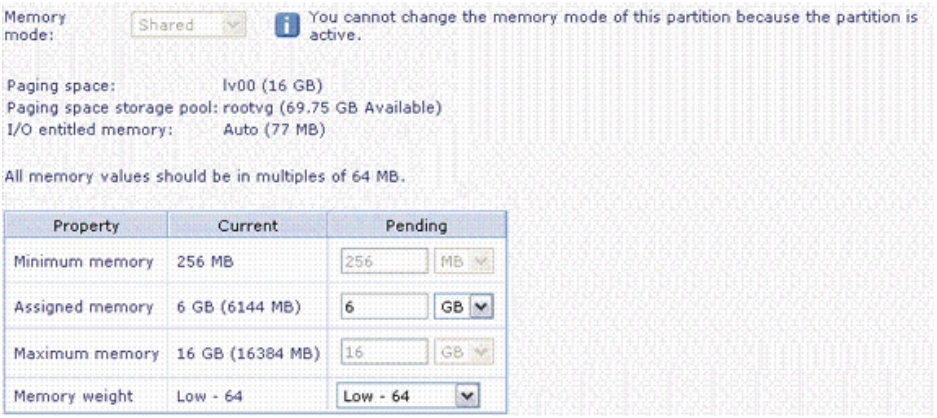
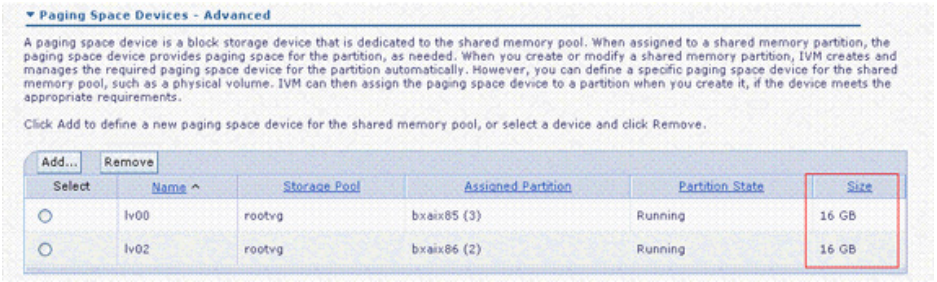


Figure 12. The LPARs' shared memory profile settings



The IVM automatically created a VIOS paging device for each LPAR as part of the migration to shared memory. It creates a paging device of equal size to that of the maximum memory value of the shared memory pool. Refer to Figure 13.

Figure 13. Paging devices created by the IVM



AMS paging devices view from the VIOS. Refer to Figure 14.

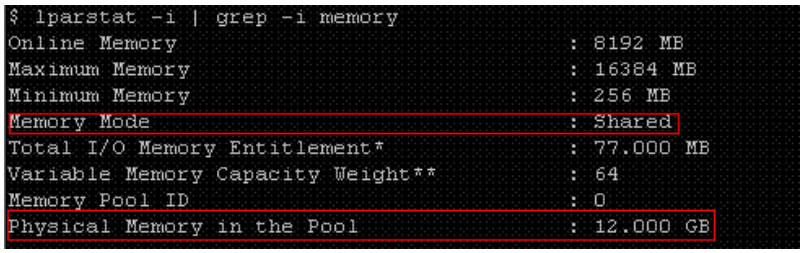
Figure 14. AMS paging devices

```
$ lsvg -lv rootvg | grep vrm
lv00               vrmdevice  64      64      1    open/syncd    N/A
lv02               vrmdevice  64      64      1    open/syncd    N/A

$ lsdev | grep vrm
vrmpage0         Available   Paging Device - Logical Volume
vrmpage2         Available   Paging Device - Logical Volume

$ lsdev -dev vrmpage0 -attr
attribute       value                 description                user_settable
LogicalUnitAddr 0x8100000000000000    Logical Unit Address       False
aix_tdev        lv00                  Target Device Name         False
redundant_usage no                    Redundant Usage            True
storage_pool    rootvg                Storage Pool               False
vasi_drc_name   U7998.61X.10071DA-V1-C15 VASI DRC Name           True
vrm_state       active                Virtual Real Memory State  True
vtd_handle      0x100016e24678d       Virtual Target Device Handle False
```

- After restarting the LPAR as a shared memory partition, I used lparstat and vmstat to view the LPARs' shared memory settings. Refer to Figures 15 and 16.

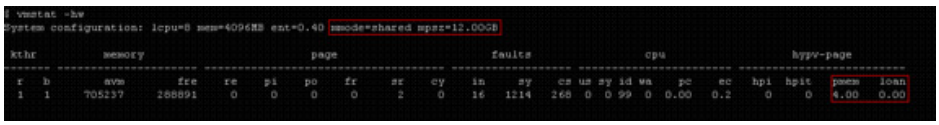
.

Figure 15. lparstat output on a shared memory LPAR



The I/O entitled memory represents the maximum amount of physical memory that is guaranteed to be available for I/O mapping by a partition at any given time. Partitions are given weight to enforce priority in allocating memory. Please refer to the IBM documentation for more information.

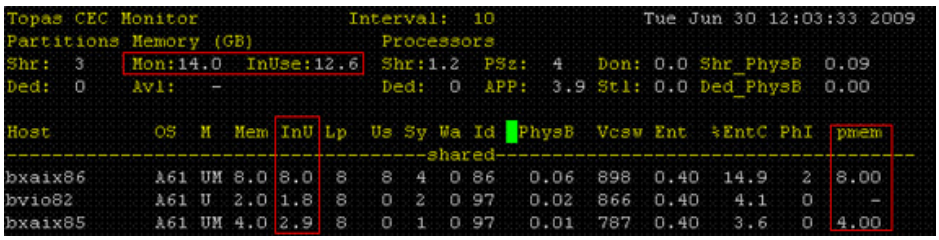
Figure 16. vmstat output on a shared memory LPAR



Monitoring AMS

Existing tools such as topas and vmstat have been enhanced to report physical memory in use, Hypervisor paging rate, Hypervisor paging rate latency, and the amount of memory loaned by AIX to the Hypervisor. These tools can be used to monitor AMS performance and activity. Refer to Figure 17.

Figure 17. topas cec output on a shared memory LPAR



pmem is the physical memory in gigabytes allocated to shared memory partitions from the shared memory pool at a given time. Please refer to the IBM documentation for more information.

The svmon tool is also AMS-aware and can display the amount of memory loaned on an LPAR. Refer to Figure 18.

Figure 18. svmon output on a shared memory LPAR

```
root@bxaix85 / # svmon -G -O unit=auto
Unit: auto
--------------------------------------------------------------------------------
               size       inuse        free         pin     virtual  available      loaned
memory         5.00G       2.96G       1.52G     493.43M       2.79G       1.52G     801.29M
pg space      20.0G        18.1M

               work       pers       clnt      other
pin          325.01M          0K         0K     440.42M
in use         2.79G          0K     177.64M
```

AMS in action

In the following example, you can observe that an LPAR requires additional memory due to an increase in workload. The memory is automatically redistributed from one LPAR to another. Memory is loaned to the busy LPAR on demand. No administrator interaction is required and the activity is transparent to the system.

Memory from bxaix85 has been loaned to bxaix86. 4GB is in use (pmem) and 2GB of memory has been loaned (loan). Refer to Figure 19.

Figure 19. Memory from bxaix85 has been loaned to bxaix86

```
kthr        memory              page                    faults        cpu             hypv-page
----- ------------------- ------------------------ ------------ ----------- ---------- -------------
 r  b       avm       fre  re  pi  po  fr  sr  cy  in   sy  cs us sy id wa  pc  ec  hpi hpit  pmem  loan
 0  0    685577    263910   0   0   0   0   0   0   6  181  256  1  2 98  0 0.02 3.8    0    0  4.00  2.00
```

Figure 20. bxaix86 is idle

```
kthr        memory              page                    faults        cpu             hypv-page
----- ------------------- ------------------------ ------------ ----------- ---------- -------------
 r  b       avm       fre  re  pi  po  fr  sr  cy  in   sy  cs us sy id wa  pc  ec  hpi hpit  pmem  loan
 0  0   1897574      2812   0   0   0   0   0   0  22  1212 1066  7  4 89  0 0.05 13.1    0    0  8.00  0.00
```

Workload starts on bxaix85. The memory that it loaned to bxaix86 is borrowed again. The amount loaned decreases (loan) and the memory in use on the LPAR (pmem) increases over time. Refer to Figure 21.

Figure 21. Workload starts on bxaix85

```
kthr        memory              page                    faults        cpu             hypv-page
----- ------------------- ------------------------ ------------ ----------- ---------- -------------
 r  b       avm       fre  re  pi  po  fr  sr  cy  in   sy  cs us sy id wa  pc  ec  hpi hpit  pmem  loan
 0  0    685577    263910   0   0   0   0   0   0   6  181  256  1  2 98  0 0.02 3.8    0    0  4.63  1.37
 .
 1  0    685576    263911   0   0   0   0   0   0   4  232  238  0  2 98  0 0.01 3.6    0    0  4.67  1.34
 .
 1  0    685575    263912   0   0   0   0   0   0   7  174  237  1  2 98  0 0.02 3.9    0    0  4.68  1.31
```

The working set of both LPARs is now larger than the shared memory pool size. Hypervisor Paging occurs (hpi and hpit) on bxaix86, as memory is given back to bxaix85. The memory in use on bxaix86 (pmem) has decreased from 8GB to just over 6GB.

Figure 22. Memory in use on bxaix86

```
kthr        memory              page                    faults        cpu             hypv-page
----- ------------------- ------------------------ ------------ ----------- ---------- -------------
 r  b       avm       fre  re  pi  po  fr  sr  cy  in   sy  cs us sy id wa  pc  ec  hpi hpit  pmem  loan
66  0   1891470      2636   0  47   0   0   0   0  56  870  731 19 26 56  0 0.18 45.2  414 1779  6.44  0.00
```

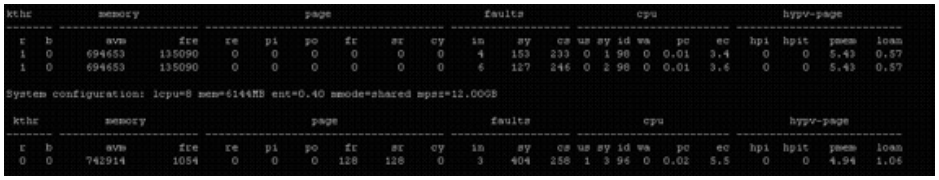
Hypervisor Paging stops on bxaix86 (hpi and hpit) once bxaix85 has enough memory to complete its work. Refer to Figure 23.

Figure 23. Hypervisor Paging stops on bxaix86

```
kthr     memory              page                    faults        cpu             hypv-page
-------- ------------------- ----------------------- ------------ ----------- ---------------------
 r  b    avm        fre      re    pi    po    fr    sr    cy    in     sy    cs us sy id wa    pc    ec  hpi  hpit  pmem  loan
 0  0    1891176    9083     0     0     0     0     0     0     35   1226  1213 10  5 86  0  0.06  16.2   12    34  6.57  0.00
 0  0    1891175    9084     0     0     0     0     0     0     38   1149  1109  9  5 86  0  0.06  15.2    0     0  6.57  0.00
```

Once bxaix85 is finished its workload, a job starts on bxaix86. Memory is, again, loaned out from bxaix85 to bxaix86. The memory in use on bxaix85 (pmem) decreases and the amount loaned (loan) increases over time. Refer to Figure 24.

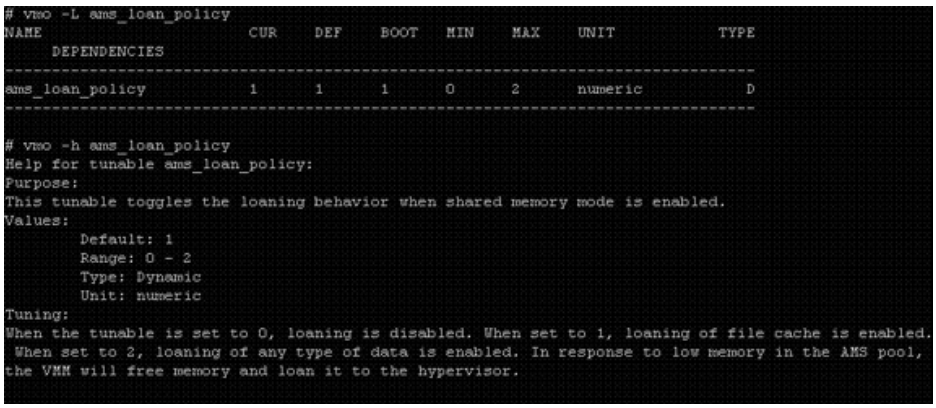
Figure 24. Memory in use on bxaix85 (pmem) decreases



```
kthr     memory              page                    faults        cpu             hypv-page
-------- ------------------- ----------------------- ------------ ----------- ---------------------
 r  b    avm        fre      re    pi    po    fr    sr    cy    in     sy    cs us sy id wa    pc    ec  hpi  hpit  pmem  loan
 1  0    694653     135090   0     0     0     0     0     0     4    153   233  0  1 98  0  0.01   3.4    0     0  5.49  0.57
 1  0    694653     135090   0     0     0     0     0     0     6    127   246  0  2 98  0  0.01   3.6    0     0  5.49  0.57

System configuration: lcpu=8 mem=6144MB ent=0.40 mmode=shared mpss=12.00GB

kthr     memory              page                    faults        cpu             hypv-page
-------- ------------------- ----------------------- ------------ ----------- ---------------------
 r  b    avm        fre      re    pi    po    fr    sr    cy    in     sy    cs us sy id wa    pc    ec  hpi  hpit  pmem  loan
 0  0    742914     1054     0     0     0     128   128   0     3    404   258  1  3 96  0  0.02   5.5    0     0  4.94  1.06
```

The AMS Loan Policy can be changed on an LPAR, using vmo. By default only file cache pages are loaned. You can change this policy, depending on how aggressive you want to be with loaning memory pages. Refer to Figure 25.

Figure 25. vmo settings on a shared memory LPAR



```
# vmo -L ams_loan_policy
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT           TYPE
     DEPENDENCIES
--------------------------------------------------------------------------------
ams_loan_policy           1      1      1      0      2      numeric          D
--------------------------------------------------------------------------------

# vmo -h ams_loan_policy
Help for tunable ams_loan_policy:
Purpose:
This tunable toggles the loaning behavior when shared memory mode is enabled.
Values:
        Default: 1
        Range: 0 - 2
        Type: Dynamic
        Unit: numeric
Tuning:
When the tunable is set to 0, loaning is disabled. When set to 1, loaning of file cache is enabled.
 When set to 2, loaning of any type of data is enabled. In response to low memory in the AMS pool,
the VMM will free memory and loan it to the hypervisor.
```

AMS is here! Now what?

The final piece of the virtualisation puzzle has arrived: Active Memory Sharing. We can let our AIX POWER systems automatically adjust memory allocation based on workload and demand. No more DLPAR operations required! By the way, DLPAR with memory is supported with AMS, so you can still add and remove memory from an LPAR dynamically, only now it's logical instead of physical memory.

There is still a lot to learn, as there will be performance tuning and monitoring differences in a virtualized memory environment. Traditional methods of monitoring AIX memory will need to be reviewed, as there are new considerations with AMS and logical memory. Just as you did with shared processing, you will, again, need to adjust your perspective on monitoring and managing virtualised resources, this time from a memory point of view.

There are still a few more areas that I need to test in my environment, such as configuring AMS with dual VIOS, performing Live Partition Mobility with AMS-enabled systems, and using AMS with

PowerHA (HACMP) clusters. I hope to find time to do this soon and with any luck report back to the AIX community. If anyone else has already done this, please let me know!

Conclusion

I hope this brief introduction to AMS gets you thinking about how you could deploy and migrate to this new technology. AMS has the potential to boost the return on investment and total cost of investment on your investment in POWER6 and PowerVM technology. Having the ability to share memory among systems will help to reduce cost and provide a more efficient method of deploying memory.

If you are going to use AMS in your environment, start thinking about how you will migrate to shared memory partitions. Start with the basics, such as:

- Updating your HMC(s) to the latest level.
- Updating your POWER6 firmware.
- Upgrading your VIO servers to version 2.1.
- Upgrading all your AIX systems to AIX 6.1.
- Producing a migration strategy for moving to AMS. Identify which systems are good candidates for AMS. For example, start with a few non-production systems. Test them until you are satisfied with the performance. Migrate more systems and repeat the process until you've virtualized as much memory as you need to.

It may also be a good time to consider some training on AMS and the latest POWER6 and AIX technologies.

Related topics

- PowerVM Virtualization Active Memory Sharing Redpaper: Introduces Active Memory Sharing on IBM Power Systems based on POWER6 Processor Technology.
- AIX6 and POWER6 Hands-On Technical Demo Movies – Look for demos on AMS concepts, setup and monitoring.