IBM ATS Technical Brief
April 2011

Oracle Database and 1 TB Segment Aliasing

Technical Brief

Ralf Schmidt-Dannert
IBM Advanced Technical Skills

Table of Contents

DISCLAIMER NOTICE

Overview

This document gives a short introduction into the new feature “1 TB Segment Aliasing” (LSA), also nick-named “Large Segment Aliasing”, and summarizes the results of a number of empirical tests illustrating how Oracle® Database works with LSA.

LSA is a new feature of AIX® 6.1 TL06 and AIX 7.1 and is supported on all hardware supported by the respective AIX release.

Two different test environments, both based on IBM Power® blades with POWER6™ processors, were utilized. The first environment had AIX 7.1 TL0 SP1 installed together with Oracle Database 11gR1. The second environment consisted of two logical partitions (LPAR) in an Oracle Real Application Clusters (Oracle RAC) configuration, utilizing the Oracle 11gR2 grid and database software stack on top of AIX 6.1 TL06 SP1.

The LSA feature is disabled by default in AIX 6.1 TL06, but enabled by default on AIX 7.1. To enable the LSA feature, set the restricted vmo parameter: ***esid_allocator***=1. For further details see section *Introduction to 1 TB Segment Aliasing*.

Quick summary if you don’t have time to read further:

- LSA works as advertised and has been tested in large proof of concepts with Oracle databases
- Oracle SGA only utilizes “Shared Aliases”
- LSA can be dynamically activated / deactivated – but the change only impacts newly started processes → DB restart required.
- LSA “trigger” values can be dynamically changed – but the change only impacts newly started processes → DB restart required.
- Oracle allocates all shared memory segments, virtual address space reservation, at startup. The virtual address space allocated is determined like this:
  - Oracle 10g:
    - `sga_target` set and `sga_max_size` not set → `sga_target`
    - `sga_max_size` set → `sga_max_size`
  - Oracle 11g:
    - `memory_target=0` and `memory_max_target` not set, see Oracle 10g
    - `memory_target>0` and `memory_max_target` not set → `memory_target`
    - `memory_max_target` set → `memory_max_target`
    - the `__sga_target` … or `__db_cache`… parameters do not seem to influence the initial virtual address space allocation.

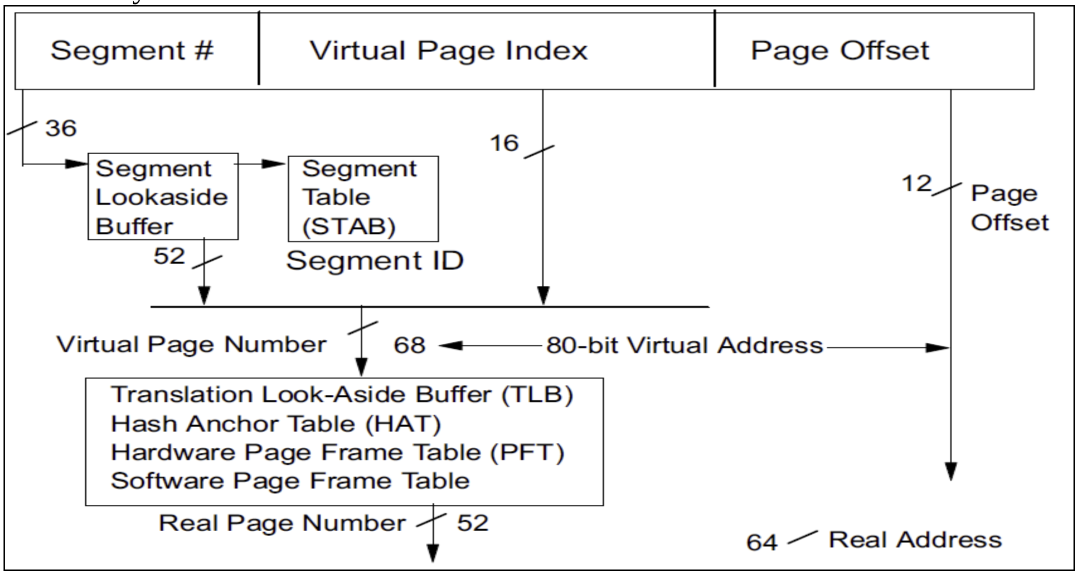
Comment:

I have not tried the per process environment variable control of the LSA feature and I have not done any comparative performance testing. I have utilized LSA in a large multi-node RAC performance test and did not encounter any stability issues. I did not investigate unshared TB segments.

Introduction to 1 TB Segment Aliasing

Source for most of the material in this section:
Saurabh Sharma, IBM: LSA presentation at T3 class in Austin in 2010:

The following graphic provides an overview how virtual addresses are normally translated into physical memory addresses in AIX on IBM Power servers.



Problem Motivation

Workloads with large memory footprints and low spatial locality perform poorly.

- Analysis has shown that Segment Lookaside Buffer (SLB) faults can take up to 20% of execution time.
- In POWER6 core, SLB has 64 entries.
  - 20 reserved for the kernel
  - 44 available for user processes → yields 11GB of accessible memory without SLB faults
- Many customer workloads do not fit into 11GB

Problem is exacerbated with POWER7 processor

- Architectural trend towards smaller SLB sizes
- POWER7 core has 32 SLB entries
  - 20 still reserved for the kernel
  - 12 available for user processes → yields 3GB of accessible memory without SLB faults
- Potential for significant performance regression.

1 TB Segment Aliasing feature allows user applications to automatically use 1TB segments.

- 12 SLB entries can now address 12TB of memory.
  - SLB fault issue no longer relevant
  - Immediate performance boost for applications, new and legacy

Significant changes under the covers

- New address space allocation policy
  - Attempts to group address space requests together to facilitate 1TB aliasing.
- Once certain allocation size thresholds have been reached, AIX automatically aliases memory with 1TB aliases.
  - 256MB segments still exist for handling IO

Note:

Aliasing is only available for shared memory regions at the time this document was written!

Shared Aliases

- A single shared memory region large enough on its own to trigger aliasing. By default, at least 3GB in size.
- TB aliases used by the entire system – “shared” by processes
- Aliasing triggered at shmat() (shared memory attach) time.
- AIX will not place other attachments into the terabyte region, unless address space pressure is present.

Unshared Aliases

- Multiple small, homogenous shared memory regions grouped into a single 1TB region.
  - Homogeneity defined as sharing same vmhandle characteristics (Ks, Kp, NLC, L, LP bits)
- Collectively large enough to trigger aliasing.
  - By default, must exceed 64GB in size.
- TB aliases are private to the process.
  - Address space layout unique to the process
- Aliasing triggered at shmat() of region that crosses threshold.
- Unshared aliasing is expensive.
  - In situ insertion of TB alias – complex serialization required.
  - shmdt() requires, that unshared alias is invalidated and removed to avoid access via page table entries (PTEs) to stale regions
  - Unshared TB alias must use brute-force PTE invalidation before being re-used.

AIX releases with 1 TB Segment Aliasing support:

- Shipped in AIX 6.1 TL 6, but off by default
- Shipped in AIX 7.1, on by default.

mmap considerations

- Aliasing not supported…yet.
- New address space layout is in use (if 1 TB Segment Aliasing active).

Unshared aliases can have performance issues!

- Every address space removal can cause an unshared alias removal.
- A 1TB unshared alias is not re-usable until a background kernel process clears all its possible Page Table Entries (PTE) up to a high-water mark established at run-time.

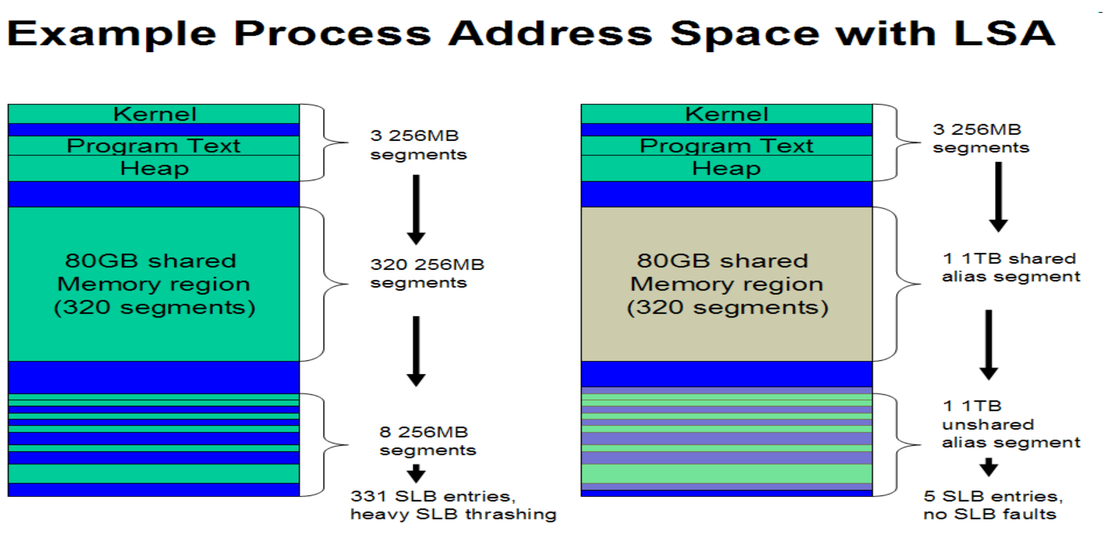

Example Process Address Space with LSA
Kernel
Program Text
Heap
80GB shared Memory region (320 segments)
3 256MB segments
320 256MB segments
8 256MB segments
331 SLB entries, heavy SLB thrashing
Kernel
Program Text
Heap
80GB shared Memory region (320 segments)
3 256MB segments
1 1TB shared alias segment
1 1TB unshared alias segment
5 SLB entries, no SLB faults

Figure 1

Four key AIX vmo parameters, all restricted, control 1 TB Segment Aliasing:

esid_allocator, VMM_CNTRL=ESID_ALLOCATOR=[0,1]
Default off (0) in AIX 6.1 TL06, on (1) in AIX 7.1. When on, indicates that 1 TB Segment Aliasing is in use including aliasing capabilities and address space layout changes. This parameter can be changed dynamically.

shm_1tb_shared, VMM_CNTRL=SHM_1TB_SHARED=[0,4096]
Default set to 12 on POWER 7, 44 on POWER 6 and earlier. Controls the threshold, "trigger value", at which a shared memory attachment will get a shared alias (3GB, 11GB). The unit is in 256MB segments.

shm_1tb_unshared, VMM_CNTRL=SHM_1TB_UNSHARED=[0,4096]
Default set to 256 (64GB). Controls the threshold at which multiple homogenous small shared memory regions will be promoted to an unshared alias. Conservatively set high as unshared aliases are compute intensive to initially establish and to destroy. The unit is in 256MB segments.

shm_1tb_unsh_enable
Default set to on; determines whether unshared aliases will be used. (Note: Unshared aliases can be "expensive")

See the following documentation for additional information on 1 TB Segment Aliasing:
http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/1TB_segment_aliasing.htm

Test Environment

- IBM JS12 blade with POWER6 processor
- Integrated Virtualization Manager + non-related LPARs deployed on the blade
- Each LPAR configured with 12GB physical memory
- AIX 7.1 TL00 SP1, Oracle single instance 11.1.0.7
- AIX 6.1 TL06 SP1, Oracle RAC 11.2.0.1 (2 LPARs)

Default settings of 1 TB Segment Aliasing related parameters in the test environment:

```
          esid_allocator = 1 (On AIX 6.1 TL06 it would be 0, disabled)
          shm_1tb_shared = 44 (On POWER7, default would be 12)
     shm_1tb_unsh_enable = 1
       shm_1tb_unshared = 256
```

Verification of 1 TB Segment Aliasing (LSA) Usage

To determine if LSA is active for a specific process you need to utilize the AIX kernel debugger “kdb” run as user “root”. To quit “kdb” enter “quit”.

Important:

Be very careful when using “kdb”, follow the described procedure carefully and make sure you typed the commands correctly, before pressing “ENTER”. You can “kill” AIX from within “kdb” if you are using the wrong commands!

Note that critical command return values are marked as “xxx” and executed commands are marked like this “yyy”. For this test “`shm_1tb_shared`” was set to 16 to force the use of LSA.
These are the necessary steps:

1) Determine PID of Oracle pmon process:

```
root@lp82 / # ps -ef|grep pmon|grep -v grep | awk '{print $2}'
  6422736

root@lp82 / # ps -fp 6422736
  UID      PID    PPID   C    STIME    TTY  TIME CMD
  oracle  6422736        1   0 17:20:47      -  0:10 ora_pmon_TST
```

2) Verify shared memory is actually allocated; in this test 32 segments were allocated based on the 32 lines with shmat/mmap description in the output from svmon.

```
root@lp82 / # svmon -P 6422736

-------------------------------------------------------------------------------
     Pid Command          Inuse      Pin     Pgsp  Virtual 64-bit Mthrd  16MB
23068902 oracle          224830     9488        0   210437      Y     N     N

     PageSize                Inuse        Pin       Pgsp      Virtual
     s    4 KB               18958          0          0         4565
     m   64 KB               12867        593          0        12867

    Vsid      Esid Type Description              PSize  Inuse   Pin Pgsp Virtual
  990fb9  70000019 work default shmat/mmap           m   2826     0    0    2826
  8a112a  7000001e work default shmat/mmap           m   1531     0    0    1531
  9810b8  70000018 work default shmat/mmap           m   1138     0    0    1138
  851185  7000001f work default shmat/mmap           m    959     0    0     959
  8e0d0e        10 clnt text data BSS heap,          s  14351     0    -       -
                     /dev/fslv00:310990
  851105  70000000 work default shmat/mmap           m    831     0    0     831
  8c108c  7000001d work default shmat/mmap           m    825     0    0     825
   20002         0 work kernel segment               m    633   590    0     633
  870e47  7000001c work default shmat/mmap           m    463     0    0     463
  960eb6  70000016 work default shmat/mmap           m    152     0    0     152
  9f105f  7000000d work default shmat/mmap           m    152     0    0     152
  9a0fda  70000017 work default shmat/mmap          m    152     0    0      152
  831083  70000010 work default shmat/mmap          m    152     0    0      152
  800fc0  70000008 work default shmat/mmap          m    152     0    0      152
  9d105d  7000000b work default shmat/mmap          m    152     0    0      152
  911091  70000001 work default shmat/mmap          m    152     0    0      152
  991039  70000004 work default shmat/mmap          m    152     0    0      152
  9a047a  70000012 work default shmat/mmap          m    152     0    0      152
  891089  70000013 work default shmat/mmap          m    152     0    0      152
  8f10af  70000011 work default shmat/mmap          m    152     0    0      152
  8e106e  70000002 work default shmat/mmap          m    152     0    0      152
  9c113c  7000000e work default shmat/mmap          m    152     0    0      152
  9210f2  7000000f work default shmat/mmap          m    152     0    0      152
  810dc1  70000003 work default shmat/mmap          m    152     0    0      152
  941074  70000007 work default shmat/mmap          m    152     0    0      152
  901090  70000009 work default shmat/mmap          m    152     0    0      152
  931073  70000005 work default shmat/mmap          m    152     0    0      152
  8b108b  70000015 work default shmat/mmap          m    152     0    0      152
  9510f5  70000006 work default shmat/mmap          m    152     0    0      152
  8c10ac  7000000c work default shmat/mmap          m    152     0    0      152
  881008  70000014 work default shmat/mmap          m    152     0    0      152
  9210b2  7000000a work default shmat/mmap          m    152     0    0      152
```

3) Now verify via kdb if the Oracle processes are supported by LSA:

```
root@lp82 / # kdb
           START              END <name>
      0000000000001000 00000000057A0000 start+000FD8
      F00000002FF47600 F00000002FFDF9C0 __ublock+000000
      000000002FF22FF4 000000002FF22FF8 environ+000000
      000000002FF22FF8 000000002FF22FFC errno+000000
      F1000F0A00000000 F1000F0A10000000 pvproc+000000
      F1000F0A10000000 F1000F0A18000000 pvthread+000000
      read vscsi_scsi_ptrs OK, ptr = 0xF1000000C01803B0
(0)>

Tip: To exit “kdb” type “quit” followed by <ENTER>
```

4) Show details for pmon process (PID from “ps” command above):

```
(0)> ppid -d 6422736
              SLOT NAME     STATE      PID     PPID          ADSPACE   CL #THS

pvproc+018800   98 oracle   ACTIVE 06200D0 0000001 0000000930D13590   0 0001

NAME....... oracle
STATE...... stat  :07  .... xstat :0000
FLAGS...... flag  :00200001 LOAD EXECED
........... flag2 :02000001 64BIT INHERITED
........... flag3 :00000000
........... atomic :00040000 ORPHANPGRP
........... secflag:0001 ROOT
LINKS...... child      :0000000000000000
........... siblings   :F1000F0A00010400 <pvproc+010400>
........... uidinfo    :F1000A18004700C0
........... ganchor    :F1000F0A00018800 <pvproc+018800>
THREAD..... threadlist :F1000F0A10013B00 <pvthread+013B00>
DISPATCH... synch      :FFFFFFFFFFFFFFFF
…
```

5) Get thread details – see "THREAD …" in output from "ppid" command

```
(0)> th F1000F0A10013B00
                SLOT NAME     STATE    TID PRI   RQ CPUID  CL  WCHAN

pvthread+013B00  315 oracle   SLEEP 13B007D 03C    0         0

NAME................ oracle
WTYPE............... WEVENT
.................tid :00000000013B007D  ......tsleep :FFFFFFFFFFFFFFFF
...............flags :00000000  ..............flags2 :00000000
...........pmcontext :00000000
DATA.........pvprocp :F1000F0A00018800 <pvproc+018800>
LINKS.....prevthread :F1000F0A10013B00 <pvthread+013B00>
..........nextthread :F1000F0A10013B00 <pvthread+013B00>
DISPATCH.......synch :FFFFFFFFFFFFFFFF
SCHEDULER...affinity :00000000  .................pri :0000003C
.............boosted :00000000  ...............wchan :0000000000000000
…
```

6) Switch to thread slot number – see value in 3rd line of output from "th" command

```
(0)> sw 315
Switch to thread: <pvthread+013B00>
```

7) Now print VMM related information for that thread. The output below shows that the pmon process is supported by LSA – see vmmflags line.

```
(0)> u -ad
User-mode address space mapping:

uadspace node allocation......(U_unode) @ F00000002FF48960
usr adspace 32bit process.(U_adspace32) @ F00000002FF48980

segment node allocation.......(U_snode) @ F00000002FF48940
segnode for 32bit process...(U_segnode) @ F00000002FF48BE0

U_adspace_lock @ F00000002FF48E20
   lock_word.....0000000000000000   vmm_lock_wait.0000000000000000
V_USERACC strtaddr:0x0000000000000000  Size:0x0000000000000000

ESID Allocator version (U_esid_allocator)........ 0001
shared alias thresh (U_shared_alias_thresh)...... 000C
unshared alias thresh (U_unshared_alias_thresh).. 0100

   vmmflags......00400507 SHMAT SHMMAP MMAP PINSHM BIGSTAB LSA_ALIAS
```

Note:

"PINSHM" is only listed if you have set "lock_sga=true" in the Oracle init.ora / spfile and given the oracle user the necessary permissions to pin the SGA memory.

Tests

1) Oracle database with SGA lower than LSA "trigger" value

I created a dummy database 'TST' and configured it with an SGA of 8GB, equivalent to 32x 256MB segments, started it up and found that "LSA_ALIAS" was not set for the pmon process. This is expected, as `shm_1tb_shared` is set to 44, which is equivalent to 11GB. So in this case, the specified "trigger" value of 44x 256MB segments for activating LSA was not met.

```
(0)> u -ad
User-mode address space mapping:

uadspace node allocation......(U_unode) @ F00000002FF48960
usr adspace 32bit process.(U_adspace32) @ F00000002FF48980

segment node allocation.......(U_snode) @ F00000002FF48940
segnode for 32bit process...(U_segnode) @ F00000002FF48BE0

U_adspace_lock @ F00000002FF48E20
   lock_word.....0000000000000000   vmm_lock_wait.0000000000000000
V_USERACC strtaddr:0x0000000000000000  Size:0x0000000000000000

ESID Allocator version (U_esid_allocator)........ 0001
shared alias thresh (U_shared_alias_thresh)...... 002C
unshared alias thresh (U_unshared_alias_thresh).. 0100

   vmmflags......00000407 SHMAT SHMMAP MMAP BIGSTAB
```

Note that LSA_ALIAS is not listed in the output!

2) Dynamic change of LSA "trigger" value without restart of Oracle database

For the next test I left the database running and just changed the `shm_1tb_shared` variable dynamically to a value of 12 and then checked the LSA settings again.

```
(0)> u -ad
User-mode address space mapping:

uadspace node allocation......(U_unode) @ F00000002FF48960
usr adspace 32bit process.(U_adspace32) @ F00000002FF48980

segment node allocation.......(U_snode) @ F00000002FF48940
segnode for 32bit process...(U_segnode) @ F00000002FF48BE0

U_adspace_lock @ F00000002FF48E20
   lock_word.....0000000000000000   vmm_lock_wait.0000000000000000
V_USERACC strtaddr:0x0000000000000000  Size:0x0000000000000000

ESID Allocator version (U_esid_allocator)........ 0001
shared alias thresh (U_shared_alias_thresh)...... 002C
unshared alias thresh (U_unshared_alias_thresh).. 0100

   vmmflags......00000407 SHMAT SHMMAP MMAP BIGSTAB
```

Note that the reflected value for the shared alias threshold is still 44 (0x2C) and does not reflect the new system wide parameter value of 12 (0xC). As the process did not see that change, the LSI_ALIAS flag is not set, implying that the process still does not utilize LSA.

3) Restart of Oracle database after dynamic change to LSA "trigger" value

Without making any other changes I now re-cycled the Oracle database and checked again:

```
(0)> u -ad
User-mode address space mapping:

uadspace node allocation......(U_unode) @ F00000002FF48960
usr adspace 32bit process.(U_adspace32) @ F00000002FF48980

segment node allocation.......(U_snode) @ F00000002FF48940
segnode for 32bit process...(U_segnode) @ F00000002FF48BE0

U_adspace_lock @ F00000002FF48E20
   lock_word.....0000000000000000   vmm_lock_wait.0000000000000000
V_USERACC strtaddr:0x0000000000000000  Size:0x0000000000000000

ESID Allocator version (U_esid_allocator)........ 0001
shared alias thresh (U_shared_alias_thresh)...... 000C
unshared alias thresh (U_unshared_alias_thresh).. 0100

   vmmflags......00400407 SHMAT SHMMAP MMAP BIGSTAB LSA_ALIAS
```

This means that an Oracle database has to be restarted to recognize a dynamic change in the `shm_1tb_shared` parameter!

4) Test to determine which database parameter actually triggers LSA

In another test, I set `sga_target`=2G and `sga_max_size`=8G and then restarted the database. The parameter `shm_1tb_shared` was still set to 12 (3GB).

```
(0)> u -ad
User-mode address space mapping:

uadspace node allocation......(U_unode) @ F00000002FF48960
usr adspace 32bit process.(U_adspace32) @ F00000002FF48980

segment node allocation.......(U_snode) @ F00000002FF48940
segnode for 32bit process...(U_segnode) @ F00000002FF48BE0

U_adspace_lock @ F00000002FF48E20
   lock_word.....0000000000000000   vmm_lock_wait.0000000000000000
V_USERACC strtaddr:0x0000000000000000  Size:0x0000000000000000

ESID Allocator version (U_esid_allocator)........ 0001
shared alias thresh (U_shared_alias_thresh)...... 000C
unshared alias thresh (U_unshared_alias_thresh).. 0100

   vmmflags......00400407 SHMAT SHMMAP MMAP BIGSTAB LSA_ALIAS
```

So the driving parameter for the number of segments Oracle 10g "reserves" is controlled in this case by the `sga_max_size` parameter and not by the `sga_target` parameter. A quick check with svmon also confirmed that oracle allocated 8G of shared memory address space in 32 segments and that in turn triggered the LSA feature. Note that this even took effect without pinning the SGA!

With Oracle 11g you can utilize Automatic Memory Management (AMM). The driving parameters for AMM are `memory_target` and `memory_max_target`.

Note:
Do not utilize `sga_target` or `sga_max_size` if you utilize AMM, as there is a good chance that you will not be able to start the database after adding/setting those parameters!

5) Test LSA in a RAC environment

In the final test, I created a RAC 11gR2 (11.2.0.1) database on AIX 6.1 TL06 SP1 with two RAC nodes. On Node 1 I activated LSA and set the limit for activation to 3GB and on Node 2, I left the AIX 6.1 TL06 SP1 defaults. The RAC database was configured with memory_target=3.5G.

Node 1:
```
(0)> u -ad
User-mode address space mapping:

uadspace node allocation......(U_unode) @ F00000002FF48960
usr adspace 32bit process.(U_adspace32) @ F00000002FF48980

segment node allocation.......(U_snode) @ F00000002FF48940
segnode for 32bit process...(U_segnode) @ F00000002FF48BE0

U_adspace_lock @ F00000002FF48E20
   lock_word.....0000000000000000   vmm_lock_wait.0000000000000000
V_USERACC strtaddr:0x0000000000000000  Size:0x0000000000000000

ESID Allocator version (U_esid_allocator)........ 0001
shared alias thresh (U_shared_alias_thresh)...... 000C
unshared alias thresh (U_unshared_alias_thresh).. 0100

   vmmflags......00440407 SHMAT SHMMAP MMAP BIGSTAB MPROT_TXT LSA_ALIAS
```

Node2:
```
(0)> u -ad
User-mode address space mapping:

uadspace node allocation......(U_unode) @ F00000002FF48960
usr adspace 32bit process.(U_adspace32) @ F00000002FF48980

segment node allocation.......(U_snode) @ F00000002FF48940
segnode for 32bit process...(U_segnode) @ F00000002FF48BE0

U_adspace_lock @ F00000002FF48E20
   lock_word.....0000000000000000   vmm_lock_wait.0000000000000000
V_USERACC strtaddr:0x0000000000000000  Size:0x0000000000000000

ESID Allocator version (U_esid_allocator)........ 0000
shared alias thresh (U_shared_alias_thresh)...... 002C
unshared alias thresh (U_unshared_alias_thresh).. 0100

   vmmflags......00040407 SHMAT SHMMAP MMAP BIGSTAB MPROT_TXT
```

As expected, on Node 1 LSA was utilized and on Node 2 it was not. If `memory_max_target` is set for the database as well, then the value of that parameter drives the initial virtual memory allocation and not `memory_target`!

Summary

1 TB Segment Aliasing works as advertised and has been tested in several large proof of concepts with Oracle databases without causing any unexpected side-effects.

In the context of Oracle databases, only the shared memory in the SGA will be supported by LSA, assuming the SGA is configured large enough to trigger 1 TB Segment Aliasing.

Some findings from the executed tests to keep in mind when working with the feature:

- LSA can be dynamically activated / deactivated – but the change only impacts newly started processes ➔ DB restart required.
- LSA "trigger" values can be dynamically changed – but the changes only impact newly started processes ➔ DB restart required.
- Oracle allocates all shared memory segments, virtual address space reservation, at startup! The virtual address space allocated is determined like this:
  - Oracle 10g:
    - `sga_target` set and `sga_max_size` not set → `sga_target`
    - `sga_max_size` set → `sga_max_size`
  - Oracle 11g:
    - `memory_target=0` and `memory_max_target` not set, see Oracle 10g
    - `memory_target>0` and `memory_max_target` not set → `memory_target`
    - `memory_max_target` set → `memory_max_target`
    - the `__sga_target` … or `__db_cache`… parameters (leading double underscore), which are added by the database into the database "spfile" at shutdown, do not seem to influence the initial virtual address space allocation.

References

- AIX 7.1 documentation library, « 1 TB Segment Aliasing » http://publib.boulder.ibm.com/infocenter/aix/v7r1/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/1TB_segment_aliasing.htm

- LSA presentation at T3 class in Austin, 2010, by Saurabh Sharma, IBM