# Table of Contents

**tests of KSM** ..... **1**
17/01/2009 ..... 1
KSM on SL5 ..... 1
installing KSM on SLC5 ..... 1
27/11/2008 ..... 1
ksm performance test ..... 1
15/11/2008 ..... 2
installing and testing ksm ..... 2
installation of latest kernel + ksm patch ..... 2
retrofitting tcmalloc ..... 4

# tests of KSM

# 17/01/2009

## KSM on SL5

### installing KSM on SLC5

KSM has been packaged as a kernel module for RedHat 5.2 (i.e. SL5). The module can be downloaded here. On a machine with kernel sources it is enough to `make, make install`. No need to reboot. W.r.t. the previous version a field in one of the control structures has changed name form `running` to `flags`: ksmcnt.c needs to be modified accordingly.

It happens that CMS acquired an Intel Core i7 and (thanks to Gilles Raimonds) SLC5 has been installed on it. (machine is `lxcmsi1`). So we just installed the module and, as usual, `chmod 666 /dev/ksm` to allow users application to interact with ksm. We may consider to modify the module itself to create this *file* with the proper access rights.

It worked out of the box and I was able to reproduce the results already presented.

# 27/11/2008

## ksm performance test

the usual cms reconstruction of cosmic event with data output. 5000 events

```
tcmalloc (ksm registering) no ksm scan
 PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  CODE DATA nFLT COMMAND
14462 innocent  20   0  944m 761m 133m R  101 19.2  27:37.41   76 697m  962 cmsRun
14466 innocent  20   0  939m 753m 133m R   98 19.0  27:11.31   76 692m  615 cmsRun
14469 innocent  20   0  938m 753m 133m R   98 19.0  27:02.18   76 692m  977 cmsRun
    cmsRun(14466) code:(p=4200, s=136360) data:(p=631844, s=64) total=772468
    cmsRun(14469) code:(p=4372, s=136364) data:(p=629892, s=64) total=770692
    cmsRun(14462) code:(p=4272, s=136372) data:(p=627348, s=64) total=768056

real   64m56.241s user   63m53.720s sys   0m28.664s
real   69m44.188s user   66m53.651s sys   0m45.347s
real   70m28.349s user   67m56.412s sys   0m45.031s

tcmalloc (ksm registering) and ksm scan
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  CODE DATA nFLT COMMAND
14953 innocent  20   0  938m 755m 558m R   98 19.1  27:00.39   76 692m    1 cmsRun
14950 innocent  20   0  940m 756m 554m R   97 19.1  27:11.45   76 693m    8 cmsRun
14946 innocent  20   0  933m 762m 559m D   75 19.2  27:36.51   76 686m    9 cmsRun
    cmsRun(14946) code:(p=4312, s=136264) data:(p=408540, s=231744) total=780860
    cmsRun(14950) code:(p=4200, s=136248) data:(p=405800, s=228104) total=774352
    cmsRun(14953) code:(p=4372, s=136256) data:(p=399700, s=233532) total=773860

real   66m10.090s user 64m1.730s sys   0m36.239s
real   71m1.074s user   67m45.936s sys   0m54.177s
real   71m1.363s user   67m20.267s sys   0m54.646s

vanilla (no tcmalloc no ksm)
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  CODE DATA nFLT COMMAND
15468 innocent  20   0 1012m 838m 117m R   99 21.1  36:47.36   76 765m  262 cmsRun
15471 innocent  20   0  999m 823m 118m R   98 20.8  36:10.90   76 753m  277 cmsRun
15475 innocent  20   0 1009m 838m 118m R   96 21.1  36:02.48   76 763m  381 cmsRun
```

```
    cmsRun(15468) code:(p=4304, s=120632) data:(p=733980, s=32) total=858948
    cmsRun(15471) code:(p=5120, s=120748) data:(p=717308, s=32) total=843208
    cmsRun(15475) code:(p=4492, s=120820) data:(p=732792, s=32) total=858136

real   68m37.440s user   67m28.632s sys   0m32.975s
real   75m14.473s user   72m19.655s sys   0m54.022s
real   75m40.310s user   73m1.902s sys   0m52.922s
```

# 15/11/2008

## installing and testing ksm

**KSM**, as described in this article on lwn by its author (or in this other article) is a linux driver that allows dynamicly sharing identical memory pages between one or more processes.

KSM scans just memory that was registered with it. Essentially this means that each memory allocation, sensible to be shared, need to be followed by a call to a *registry function*.

I used for my test `lxcmsh1` a 4core amd **Dual-Core AMD Opteron Processor 2216** running fedora 8.

### installation of latest kernel + ksm patch

untar the attached set of kernel patches

- ksm.tar.bz2: ksm_v2 patches

```
sudo git clone git:/git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git linux-2.6

[innocent@lxcmsh1 linux-2.6.ksm_v2]$ sudo patch -p1 < ~/ksm_v_2_/0001-Rmap-Add-page_wrprotect-fun
patching file include/linux/rmap.h
patching file mm/rmap.c
[innocent@lxcmsh1 linux-2.6.ksm_v2]$ sudo patch -p1 < ~/ksm_v_2_/0002-Add-replace_page-change-the
patching file include/linux/mm.h
patching file mm/memory.c
[innocent@lxcmsh1 linux-2.6.ksm_v2]$ sudo patch -p1 < ~/ksm_v_2_/0003-add-ksm-kernel-shared-memor
patching file include/linux/ksm.h
patching file include/linux/miscdevice.h
patching file mm/Kconfig
patching file mm/Makefile
patching file mm/ksm.c
[innocent@lxcmsh1 linux-2.6.ksm_v2]$ sudo patch -p1 < ~/ksm_v_2_/0004-MMU_NOTIFIRES-add-set_pte_a
patching file arch/x86/include/asm/kvm_host.h
patching file arch/x86/kvm/mmu.c
patching file include/linux/mmu_notifier.h
patching file mm/memory.c
patching file mm/mmu_notifier.c
patching file virt/kvm/kvm_main.c

sudo make -j 4
sudo make modules_install
sudo make install
"sh /usr/src/linux-2.6.ksm_v2/arch/x86/boot/install.sh 2.6.28-rc4 arch/x86/boot/bzImage System.ma

sudo cp /usr/src/linux-2.6.ksm/include/linux/ksm.h /usr/include/linux/.

# now edit /etc/grub.conf
sudo reboot
name -a
Linux lxcmsh1 2.6.28-rc4 #2 SMP Tue Nov 18 08:57:44 CET 2008 x86_64 x86_64 x86_64 GNU/Linux
```

```
# after each reboot
sudo chmod 666 /dev/ksm
```

compile and lik the ksm controller

- ksmcnt.c: ksm controller

start the ksm thread (with reasonable parameters...)

```
./ksmcnt start 100000 100000 10000000
./ksmcnt info
```

With these values ksm with scan, and eventually merge, 400MB worth of memory each 10 seconds. This is useful at beginning to speed up the sharing of all pages, but its surely too heavy at steady state. Therefore a more dynamical control of ksm is required. For instance scanning and merging 50MB each minutes or so can be a reasonable value for slowly changing *conditions*.

compile the registration function to be invoked in the user application

- ksm_register.c: function to register memory to ksm

As test I used a simple c++ program

- verysimpleMG.cpp: simple test of kernel memory sharing

compiled as `c++ -O2  verysimpleMG.cpp ksm_register.o -o vsimpleMG`

that allocates a vector of vectors of floats (1000*50000) registering each vector independently. They are all copied in a second vector that is modified in a loop. In a second loop only a part of (in the run below half of each `vector`) the second vector is modified.

the result shown by top and the `smap-dump` are fully consistent with the expectations...

```
 7860 innocent  20   0  394m 383m 188m R  100  9.7   0:51.70 vsimpleMG
 7861 innocent  20   0  394m 383m 188m R  100  9.7   0:50.70 vsimpleMG
 7862 innocent  20   0  394m 383m 188m R  100  9.7   0:48.62 vsimpleMG
 7863 innocent  20   0  394m 383m 188m R  100  9.7   0:49.65 vsimpleMG

------------------------------------------------------------

 7861 innocent  20   0  394m 383m 282m R  100  9.7   4:12.88 vsimpleMG
 7862 innocent  20   0  394m 383m 297m R  100  9.7   4:13.78 vsimpleMG
 7863 innocent  20   0  394m 383m 287m R  100  9.7   4:14.52 vsimpleMG
 7860 innocent  20   0  394m 383m 282m R   95  9.7   4:16.20 vsimpleMG
  228 root      15  -5     0          0         0     R       5  0.0   5:01.59 kksmd
```

I've then moved to a more complex test replacing the `float` with a class with virtual table and non trivial attribute members such as pointers and process specific values.

- memoryGame.cpp: memoryGame.cpp

Again KSM behaves as expected, as long as the memory content is identical in all processes the pages are shared. As soon as the memory content differs, either because the memory allocation pattern in two processes is different (and therefore the pointer to the *same* object has not the same value) or because some value depends on the process itself the pages remain private.

### retrofitting tcmalloc

at this point I decided to retrofic tcmalloc (see below). It turns out that it is enough to modify just one function in `tcmalloc.cc` adding just one line

```
ool TCMalloc_PageHeap::GrowHeap(Length n) {
  ASSERT(kMaxPages >= kMinSystemAlloc);
  if (n > kMaxValidPages) return false;
  Length ask = (n>kMinSystemAlloc) ? n : static_cast<Length>(kMinSystemAlloc);
  size_t actual_size;
  void* ptr = TCMalloc_SystemAlloc(ask << kPageShift, &actual_size, kPageSize);
  if (ptr == NULL) {
    if (n < ask) {
      // Try growing just "n" pages
      ask = n;
      ptr = TCMalloc_SystemAlloc(ask << kPageShift, &actual_size, kPageSize);
    }
    if (ptr == NULL) return false;
  }
  ask = actual_size >> kPageShift;
  RecordGrowth(ask << kPageShift);
  ksm_register_memory((char*)(ptr), actual_size);
```

at this point just the usual installation (32bit!)

and got running three identical `cmsrun jobs` (tracker reconstruction of simulated data, no output)

```
30171 innocent  20   0  680m 517m 364m R  100 13.0   3:03.48 cmsRun
30214 innocent  20   0  672m 508m 328m R  100 12.8   1:26.49 cmsRun
30216 innocent  20   0  673m 510m 348m R  100 12.9   1:32.42 cmsRun

    cmsRun(30214) code:(p=0, s=112976) data:(p=216800, s=193868) total=523644
    cmsRun(30216) code:(p=0, s=112976) data:(p=213328, s=198284) total=524588
    -bash(13738) code:(p=0, s=1164) data:(p=424, s=104) total=1692
    -bash(12780) code:(p=32, s=1180) data:(p=676, s=104) total=1992
    cmsRun(30171) code:(p=12, s=112976) data:(p=229780, s=186764) total=529532
```

while w/o the `LD_PRELOAD` of tcmalloc it stays as

```
30286 innocent  20   0  678m 522m 110m R  100 13.2   2:17.79 cmsRun
30310 innocent  20   0  438m 289m 105m R   76  7.3   1:04.68 cmsRun
30308 innocent  20   0  434m 286m 104m R   72  7.2   1:00.66 cmsRun


    cmsRun(30308) code:(p=0, s=109572) data:(p=273036, s=64) total=382672
    cmsRun(30310) code:(p=0, s=109564) data:(p=263952, s=64) total=373580
    -bash(13738) code:(p=0, s=1164) data:(p=424, s=104) total=1692
    -bash(12780) code:(p=32, s=1180) data:(p=676, s=104) total=1992
    cmsRun(30286) code:(p=3348, s=109572) data:(p=429332, s=64) total=542316
```

similar running on three different files:

```
    cmsRun(2434) code:(p=0, s=112976) data:(p=286872, s=125488) total=525336
    cmsRun(2436) code:(p=12, s=112976) data:(p=277752, s=127648) total=518388
    cmsRun(2438) code:(p=16, s=108316) data:(p=223512, s=112324) total=444168
```

same running full reconstruction of real cosmic data (with root-tree output)

```
    cmsRun(2604) code:(p=4272, s=135832) data:(p=360512, s=231436) total=732052
    cmsRun(2608) code:(p=4208, s=136144) data:(p=364984, s=230072) total=735408
    cmsRun(2611) code:(p=4208, s=136180) data:(p=363940, s=230640) total=734968
----
 2604 innocent  20   0  915m 720m 534m R  100 18.2  11:10.02 cmsRun
```

```
 2611 innocent  20   0  921m 724m 528m R   98 18.2  11:32.85 cmsRun
 2608 innocent  20   0  923m 726m 526m R   97 18.3  11:15.07 cmsRun
```

After one hour of running things have evolved a bit (not for what data sharing is concerned though)

```
    cmsRun(2604) code:(p=4260, s=85244) data:(p=408968, s=261128) total=759600
    cmsRun(2608) code:(p=4232, s=85852) data:(p=403724, s=255112) total=748920
    cmsRun(2611) code:(p=11664, s=85804) data:(p=405028, s=255160) total=757656
......
 2604 innocent  20   0  955m 733m 567m R  101 18.5  66:57.96 708m    0 cmsRun
 2608 innocent  20   0  953m 731m 551m R   99 18.4  65:54.74 706m    0 cmsRun
 2611 innocent  20   0  953m 740m 558m R   99 18.7  66:01.01 707m    0 cmsRun
```

(new columns are total-data and dirty pages...)

to those interested after three hours of running, 14K events processed and about 3.5GB of output written by each process

```
    cmsRun(2990) code:(p=4204, s=45572) data:(p=419560, s=251588) total=720924
    cmsRun(2993) code:(p=15920, s=48872) data:(p=419448, s=253676) total=737916
    cmsRun(2996) code:(p=4964, s=48492) data:(p=421352, s=253604) total=728412
----
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  DATA nFLT nDRT COMMAND
 2990 innocent  20   0  959m 704m 550m R   99 17.7 183:59.74 712m  844    0 cmsRun
 2993 innocent  20   0  960m 720m 558m R   98 18.2 180:40.91 714m 1452    0 cmsRun
 2996 innocent  20   0  959m 711m 545m R   97 17.9 180:05.54 713m 1005    0 cmsRun
```

same job **W/O** output module

```
    cmsRun(3801) code:(p=4208, s=135108) data:(p=302532, s=226192) total=668040
    cmsRun(3804) code:(p=4200, s=135016) data:(p=305884, s=227504) total=672604
    cmsRun(3807) code:(p=4384, s=135100) data:(p=306160, s=224976) total=670620
------------
 PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  DATA nFLT nDRT COMMAND
 3801 innocent  20   0  800m 652m 495m R  100 16.5   7:30.43 553m  890    0 cmsRun
 3804 innocent  20   0  805m 657m 495m R   99 16.6   7:18.58 559m  982    0 cmsRun
 3807 innocent  20   0  805m 654m 495m R   97 16.5   7:17.25 559m  955    0 cmsRun
  227 root      15  -5     0    0    0 S   43  0.0  61:22.40    0    0    0 kksmd
```

which confirms that, at least for CMS data model, 100MB of unshared data are used just to write in the root tree

For completeness I stopped KSM for a while and so that some 30MB slowly becoming private. They come back *shared* as soon as KSM is started again. The origin of these fluctuation are not clear to me: maybe they are just accidental identical pages (full of zeros?).

```
stopping KSM for 30 minutes
    cmsRun(9508) code:(p=4268, s=132660) data:(p=440300, s=211912) total=789140
    cmsRun(9511) code:(p=4204, s=132752) data:(p=434112, s=222516) total=793584
    cmsRun(9514) code:(p=4372, s=132684) data:(p=434260, s=223680) total=794996
few minutes after restarting KSM
    cmsRun(9508) code:(p=4280, s=132636) data:(p=409556, s=242808) total=789280
    cmsRun(9511) code:(p=4204, s=132728) data:(p=400788, s=254112) total=791832
    cmsRun(9514) code:(p=4372, s=132660) data:(p=399132, s=257168) total=793332
```

-- VincenzoInnocente - 02-Mar-2011

---

This topic: LCG > VIKSM
Topic revision: r1 - 2011-03-02 - unknown