
Subject: [RFC][-mm PATCH 0/8] Memory controller introduction (v3)

Posted by [Balbir Singh](#) on Fri, 20 Jul 2007 08:23:52 GMT

[View Forum Message](#) <> [Reply to Message](#)

Changelog since version 2

1. Improved error handling in mm/memory.c (spotted by YAMAMOTO Takashi)
2. Test results included
3. `try_to_free_mem_container_pages()` bug fix (`sc->may_writepage` is now set to `!laptop_mode`)

Changelog since version 1

1. Fixed some compile time errors (in mm/migrate.c from Vaidyanathan S)
2. Fixed a panic seen when `LIST_DEBUG` is enabled
3. Added a mechanism to control whether we track page cache or both page cache and mapped pages (as requested by Pavel)

This patchset implements another version of the memory controller. These patches have been through a big churn, the first set of patches were posted last year and earlier this year at <http://lkml.org/lkml/2007/2/19/10>

This patchset draws from the patches listed above and from some of the contents of the patches posted by Vaidyanathan for page cache control. <http://lkml.org/lkml/2007/6/20/92>

At OLS, the resource management BOF, it was discussed that we need to manage RSS and unmapped page cache together. This patchset is a step towards that

TODO's

1. Add memory controller water mark support. Reclaim on high water mark
2. Add support for shrinking on limit change
3. Add per zone per container LRU lists (this is being actively worked on by Pavel Emelianov)
4. Make `page_referenced()` container aware
5. Figure out a better CLUI for the controller
6. Add better statistics

In case you have been using/testing the RSS controller, you'll find that this controller works slower than the RSS controller. The reason being that both swap cache and page cache is accounted for, so pages do go out to swap upon reclaim (they cannot live in the swap cache).

I've test compiled the framework without the controller enabled, tested the code minimally on a power box.

Any test output, feedback, comments, suggestions are welcome!

NOTE: We use `css_put()`, container API. This API is going to change and soon become non-blocking. If you face any problem running this container, please do report it. We expect the new changes to solve problems. To be on the safer side, you could not enable `notify_on_release` for the memory controller container (which is the default behaviour right now).

Using the patches

1. Enable Memory controller configuration
2. Compile and boot the new kernel
3. `mount -t container container -o mem_container /container`
will mount the memory controller to the `/container` mount point
4. `mkdir /container/a`
5. `echo $$ > /container/a/tasks` (add tasks to the new container)
6. `echo -n <num_pages> > /container/a/mem_limit`
example
`echo -n 204800 > /container/a/mem_limit`, sets the limit to 800 MB
on a system with 4K page size
7. run tasks, see the memory controller work
8. Report results, provide feedback
9. Develop/use new patches and go to step 1

Test Results

Kernbench (run with `kernbench -M`)

Kernbench did not notice any significant slow (or speed up :-)) down due to the memory controller patches

Container not limited

Average Half Load -j 3 Run:

Elapsed Time 902.79

User Time 845.43

System Time 77.378

Percent CPU 102

Context Switches 33438.6

Sleeps 125006

Average Optimal -j 16 Load Run:

Elapsed Time 386.56

User Time 684.058

System Time 51.86

Percent CPU 189.8

Context Switches 249568
Sleeps 201478

Container limited to 800 MB

Average Half Load -j 3 Run:
Elapsed Time 902.036
User Time 844.96
System Time 77.28
Percent CPU 102
Context Switches 34082.8
Sleeps 125713

Average Optimal -j 16 Load Run:
Elapsed Time 386.66
User Time 683.962
System Time 51.834
Percent CPU 189.6
Context Switches 246458
Sleeps 198505

Container limited to 400 MB

Average Half Load -j 3 Run:
Elapsed Time 903.892
User Time 845.002
System Time 77.436
Percent CPU 101.6
Context Switches 32939.4
Sleeps 124572

Average Optimal -j 16 Load Run:
Elapsed Time 387.084
User Time 684.064
System Time 51.9
Percent CPU 189.6
Context Switches 250638
Sleeps 203246

Memory controller not compiled in

Average Half Load -j 3 Run:
Elapsed Time 888.548
User Time 840.758
System Time 66.274

Percent CPU 101.6
Context Switches 34090.8
Sleeps 125883

Average Optimal -j 16 Load Run:
Elapsed Time 378.514
User Time 677.34
System Time 45.41
Percent CPU 190.6
Context Switches 244298
Sleeps 195483

Lmbench

Legend

800 ==> Container limited to using 800 MB of memory
disabled ==> Memory controller not compiled in
enabled ==> Memory controller compiled in but not mounted
unlimited ==> Memory controller compiled in and mounted, no memory limit
was set

The interesting data in the test comes from

- (1) fork and exec processing times
- (2) Page fault and mmap latency

L M B E N C H 2 . 0 S U M M A R Y

Basic system parameters

Host	OS	Description	Mhz
800	Linux 2.6.22-	x86_64-linux-gnu	3597
disabled	Linux 2.6.22-	x86_64-linux-gnu	3597
enabled	Linux 2.6.22-	x86_64-linux-gnu	3597
unlimited	Linux 2.6.22-	x86_64-linux-gnu	3597

Processor, Processes - times in microseconds - smaller is better

Host	OS	Mhz	null	null	open	selct	sig	sig	fork	exec	sh
			call	I/O	stat	clos	TCP	inst	hndl	proc	proc
800	Linux 2.6.22-	3597	0.18	0.26	1.28	2.29	5.577	0.32	2.97	105.	500. 6351

disabled Linux 2.6.22- 3597 0.18 0.25 1.26 2.29 5.578 0.33 2.96 96.1 436. 6155
 enabled Linux 2.6.22- 3597 0.18 0.26 1.27 2.15 5.587 0.32 2.96 105. 469. 6380
 unlimited Linux 2.6.22- 3597 0.18 0.26 1.27 2.27 5.580 0.32 2.96 105. 498. 6358

Context switching - times in microseconds - smaller is better

Host	OS	2p/0K ctxsw	2p/16K ctxsw	2p/64K ctxsw	8p/16K ctxsw	8p/64K ctxsw	16p/16K ctxsw	16p/64K ctxsw
800	Linux 2.6.22-	7.620	8.6400	10.6	8.4600	14.3	9.10000	17.5
disabled	Linux 2.6.22-	7.430	8.2600	11.0	8.4600	15.0	10.2	18.9
enabled	Linux 2.6.22-	6.790	8.3900	10.7	9.8300	15.8	10.6	17.2
unlimited	Linux 2.6.22-	7.610	8.4700	10.6	10.7	14.9	10.9	17.9

Local Communication latencies in microseconds - smaller is better

Host	OS	2p/0K ctxsw	Pipe UNIX	AF UDP	UDP UDP	RPC/ TCP conn	TCP TCP	RPC/ TCP	TCP
800	Linux 2.6.22-	7.620	16.8	15.9		26.9	32.4		
disabled	Linux 2.6.22-	7.430	17.3	14.3		27.3	32.4		
enabled	Linux 2.6.22-	6.790	14.9	16.8		27.0	48.7		
unlimited	Linux 2.6.22-	7.610	16.7	17.4		27.1	33.3		

File & VM system latencies in microseconds - smaller is better

Host	OS	0K File Create	10K File Delete	Mmap Create	Prot Delete	Page Latency	Page Fault	Page Fault
800	Linux 2.6.22-	16.1	9.8000	50.0	23.7	2876.0	0.657	2.00000
disabled	Linux 2.6.22-	16.3	9.9090	51.8	22.8	5475.0	0.625	1.00000
enabled	Linux 2.6.22-	16.0	9.6860	49.5	23.5	6616.0	0.641	2.00000
unlimited	Linux 2.6.22-	16.1	9.7350	49.8	23.6	6629.0	0.646	2.00000

Local Communication bandwidths in MB/s - bigger is better

Host	OS	Pipe UNIX	AF reread	TCP reread	File reread (libc)	Mmap (hand)	Bcopy read	Bcopy write	Mem	Mem
800	Linux 2.6.22-	1078	468.	911.	2287.2	3665.4	1106.8	1055.0	3658	1599.
disabled	Linux 2.6.22-	1034	416.	917.	2282.9	3608.4	1113.2	1057.2	3602	1612.
enabled	Linux 2.6.22-	1077	419.	909.	2285.7	3616.4	1113.8	1066.1	3623	1614.
unlimited	Linux 2.6.22-	1064	415.	912.	2282.7	3610.6	1108.9	1053.6	3611	1599.

Memory latencies in nanoseconds - smaller is better
 (WARNING - may not be correct, check graphs)

Host	OS	Mhz	L1 \$	L2 \$	Main mem	Guesses
------	----	-----	-------	-------	----------	---------

800 Linux 2.6.22- 3597 1.111 7.7940 56.7
disabled Linux 2.6.22- 3597 1.111 7.8470 57.8
enabled Linux 2.6.22- 3597 1.111 7.8410 57.7
unlimited Linux 2.6.22- 3597 1.111 7.8350 57.8

series

res_counters_infra.patch
mem-control-setup.patch
mem-control-accounting-setup.patch
mem-control-accounting.patch
mem-control-task-migration.patch
mem-control-lru-and-reclaim.patch
mem-control-out-of-memory.patch
mem-control-choose-rss-vs-rss-and-pagecache.patch

--

Warm Regards,
Balbir Singh
Linux Technology Center
IBM, ISTL

Containers mailing list
Containers@lists.linux-foundation.org
<https://lists.linux-foundation.org/mailman/listinfo/containers>
