Virtual memory settings in Linux - The Problem with Overcommit

How to tune the Memory Overcommit settings in Linux

Posted on Sat, Jul 2, 2016 by Andreas Scherbaum (/authors/ascherbaum)

Categories: Linux (/categories/linux) Greenplum Database (/categories/greenplum-database) Virtual Memory (/categories/virtual-memory) Overcommit (/categories/overcommit)

Edit this post on GitHub. (https://github.com/pivotal-cf/blog/edit/master/content/post/Virtual_memory_settings_in_Linux_-_The_problem_with_Overcommit.md)

Tweet (https://twitter.com/share)

Greenplum Database (http://greenplum.org/) users sometimes add more RAM to their segment servers, but forget to adapt the Memory Overcommit settings to make use of all the new memory. After such an upgrade, at first glance, not much happens - except that the system is not using all the memory. But unless you take a close look at the memory usage on the segment hosts, you will not see that some memory goes unused.

To understand what's going on, let's first dive into the Linux Memory Management and what Overcommit actually means. Here is the background story:

Background

RAM & Swap

In modern operating systems like Linux, you often have two types of memory available: physical memory (RAM (https://en.wikipedia.org/wiki/Random_Access_Memories)) and on-disk memory (Swap (https://en.wikipedia.org/wiki/Paging#Terminology)). Back in the old days where memory was really expensive, someone came up with the clever idea to page out currently unused memory pages to disk. If the program which allocated the memory page in the first place needed to access the page again, it was loaded from disk - and possibly other currently unused pages were swapped out in order to make some space in memory. Obviously this process is slow. Modern DRAM has access times of less than 100 Nanoseconds, where a typical spinning disk drive needs a few Milliseconds to access a block on disk. Writing out a page is even slower, and if the system is already under heavy load, other processes might require access to the precious I/O channels as well.

Nevertheless using Swap gives the advantage of having more memory available, just in case an application needs momentarily more RAM than physically available. However for a database system it severely degrades the performance, if the data needs to be read from disk, then swapped out, and swapped in again. It is best practice not to use swap at all on a dedicated database server.

Overcommiting Memory

In an ideal world, every application would only request as much memory as it currently needs. And frees memory instantly when it is no longer used.

Unfortunately that is not the case in the real world. Many applications request more memory - just in case. Or preallocate memory which in the end is never used. Ever heard of *Java Heap Space*? Or a webbrowser eating Gigabytes of memory, even though most websites are already closed? Here you go …

Fortunately the Operating System knows about the bad habits of applications, and overcommits memory. That is, it provisions more memory (both RAM and Swap) than it has available, betting on applications to reserve more memory pages than they actually need. Obviously this will end in an Out-of-Memory disaster if the application really needs the memory, and the kernel will go around and kill applications. The error message you see is similar to "Out of memory: Kill process …", or in short: "OOM-Killer" (https://www.hawatel.com/blog/kernel-out-of-memory-kill-process). To avoid such situations, the overcommit behavior is configurable.

The two parameters (https://www.kernel.org/doc/Documentation/sysctl/vm.txt) to modify the overcommitt settings are */proc/sys/vm/overcommit_memory* and */proc/sys/vm/overcommit_ratio*.

/proc/sys/vm/overcommit_memory

This switch knows 3 different settings (https://www.kernel.org/doc/Documentation/vm/overcommit-accounting):

- **0**: The Linux kernel is free to overcommit memory (this is the default), a heuristic algorithm is applied to figure out if enough memory is available.
- **1**: The Linux kernel will always overcommit memory, and never check if enough memory is available. This increases the risk of out-of-memory situations, but also improves memory-intensive workloads.
- **2**: The Linux kernel will not overcommit memory, and only allocate as much memory as defined in *overcommit_ratio*.

The setting can be changed by a superuser:

```
echo 2 > /proc/sys/vm/overcommit_memory
```

/proc/sys/vm/overcommit_ratio

This setting is only used when *overcommit_memory = 2*, and defines how many percent of the physical RAM are used. Swap space goes on top of that. The default is "50", or 50%.

The setting can be changed by a superuser:

```
echo 75 > /proc/sys/vm/overcommit_ratio
```

Linux Memory Allocation

For the purpose of this post we mainly look into *overcommit_memory = 2*, we don't want to end up in situations where a process is killed by the OOM-killer.

Linux uses a simple formula to calculate how much memory can be allocated:

```
Memory Allocation Limit = Swap Space + RAM * (Overcommit Ratio / 100)
```

Let's look at some some numbers:

Scenario 1:

4 GB RAM, 4 GB Swap, overcommit_memory = 2, overcommit_ratio = 50

- Memory Allocation Limit = 4 GB Swap Space + 4 GB RAM * (50% Overcommit Ratio / 100)
- Memory Allocation Limit = 6 GB

Scenario 2:

4 GB RAM, 8 GB Swap, overcommit_memory = 2, overcommit_ratio = 50

- Memory Allocation Limit = 8 GB Swap Space + 4 GB RAM * (50% Overcommit Ratio / 100)
- Memory Allocation Limit = 10 GB

Scenario 3:

4 GB RAM, 2 GB Swap, overcommit_memory = 2, overcommit_ratio = 50

- Memory Allocation Limit = 2 GB Swap Space + 4 GB RAM * (50% Overcommit Ratio / 100)
- Memory Allocation Limit = 4 GB

Note that this is the total amount of memory which Linux will allocate. This includes all running daemons and other applications. Don't assume that your application will be able to allocate the total limit. Linux will also provide the memory allocation limit in the field *CommitLimit* in */proc/meminfo*.

In order to verify these settings and findings, I created a virtual machine and ran tests with different *overcommit_memory* and *overcommit_ratio* settings. A small program (http://stackoverflow.com/questions/911860/does-malloc-lazily-create-the-backing-pages-for-an-allocation-on-linux-and-othe) is used to allocate memory in 1 MB steps, until it fails because no more memory is available.

4 GB RAM, 4 GB Swap, overcommit_memory = 2:

overcommit_ratio	MemFree (kB)	CommitLimit (kB)	Breaks at (MB)	Expected break (MB)	Diff expected and actual break (MB)
10	3803668	4595144	4200	4488	68
25	3802056	5199488	4793	5078	63
50	3801852	6206724	5771	6062	69
75	3802732	7213960	6748	7045	76
90	3802620	7818300	7340	7636	75
100	3802888	8221196	7729	8029	79

- *overcommit_ratio* shows the setting in /proc/sys/vm/overcommit_ratio (the VM was rebooted after every test)
- *MemFree* shows the free memory right before the test was started
- *CommitLimit* shows the entry in */proc/meminfo*
- *Breaks at* shows how much memory the program was able to allocate
- *Expected break* is the above calculation again, and should show the same number as *CommitLimit*, just in MB
- *Difference expected and actual break* shows the difference between the expected break and the actual break, but takes *MemFree* into account - that is, it calculates how much memory the test application could possibly allocate based on the free memory before running the test

As you can see, the difference between what the program is expected to allocate based on the free memory, and the actual number is small. Take into account that the program itself also needs some memory for the code, loaded libraries, stack and such.

Also important: **Using the default settings the system will use around 6 GB of combined RAM and Swap.**

Let's look at two more examples.

4 GB RAM, no Swap, overcommit_memory = 2:

overcommit_ratio	MemFree (kB)	CommitLimit (kB)	Breaks at (MB)	Expected break (MB)	Diff expected and actual break (MB)
10	3803964	402892	243	394	69
25	3802532	1007236	803	984	40
50	3799844	2014472	1756	1968	12
75	3803580	3021708	2708	2951	23
90	3805424	3626048	3276	3542	48
100	3804236	4028944	3653	3935	63

If you leave the *overcommit_ratio = 50*, the test application can only allocate half the memory (50%). Technically, **all applications and daemons on the system can only use 2 GB RAM altogether**. Only if *overcommit_ratio* is changed to *100* in this scenario, the entire memory is used.

8 GB RAM, 4 GB Swap, overcommit_memory = 2:

overcommit_ratio	MemFree (kB)	CommitLimit (kB)	Breaks at (MB)	Expected break (MB)	Diff expected and actual break (MB)
10	7921080	5008020	4607	4891	53
25	7922144	6231680	5797	6086	59
50	7920776	8271108	7784	8078	63
75	7922520	10310536	9758	10069	81
90	7922820	11534192	10946	11264	89
100	7921180	12349964	11747	12061	83

The memory is doubled, and now the application will use around 8 GB RAM if *overcommit_ratio* is set to 50%.

Two more examples, this time with different overcommit_memory settings:

4 GB RAM, 4 GB Swap, overcommit_memory = 0:

overcommit_ratio	MemFree (kB)	CommitLimit (kB)	Breaks at (MB)
10	3802128	4595144	7802
25	3800896	5199488	7806
50	3802776	6206724	7794
75	3798900	7213960	7805
90	3804468	7818300	7808
100	3803396	8221196	7798

4 GB RAM, 4 GB Swap, overcommit_memory = 1:

overcommit_ratio	MemFree (kB)	CommitLimit (kB)	Breaks at (MB)
10	3803952	4595144	7794

overcommit_ratio	MemFree (kB)	CommitLimit (kB)	Breaks at (MB)
25	3804852	5199488	7804
50	3804564	6206724	7800
75	3805032	7213960	7800
90	3802900	7818300	7803
100	3801392	8221196	7748

In both scenarios the OS will allocate as much memory as possible, even using swap - which is a no-go for a database server. That is the reason why *overcommit_memory* should always be set to *2* if you run a dedicated server.

Running several applications in parallel

One last example, showing that the OS is taking all applications into account when calculating the memory for our test application:

Host memory is 8 GB, overcommit_ratio = 100, overcommit_memory = 2. Same scenario as the third table, last line. This time, two applications are running: one is allocating 4 GB of memory, then the other application is started.

The test application breaks at 7641 MB. This number plus the 4096 MB from the other application together is 11737 MB - just 10 MB difference from the 11747 MB in the last test in the table.

Conclusions

If you expand the memory in a system, always check the *overcommit_ratio* setting. Else you will end up not using all the memory you just spent money on, or even worse your server will swap memory pages to disk.

Here is the formula for not using Swap, but using all RAM:

```
Overcommit Ratio = 100 * ((RAM - Swap Space) / RAM)
```

EMC DCA

EMC (https://www.emc.com/) offers a Data Computing Appliance (https://pivotal.io/big-data/emc-dca) (DCA), which is a preconfigured Greenplum Database (http://greenplum.org/) system running on commodity hardware. The v2 of the DCA, by default, has 64 GB RAM and 32 GB Swap on every server:

```
[gpadmin@seg1 ~]$ cat /proc/meminfo
MemTotal:     65886676 kB (64342 MB)
SwapTotal:    33553400 kB (32766 MB)
```

Let’s do the math:

```
Overcommit Ratio = 100 * ((64 GB - 32 GB) / 64 GB)
Overcommit Ratio = 50
```

And sure enough:

```
[gpadmin@seg1 ~]$ cat /proc/sys/vm/overcommit_ratio
50
```

That's fine. Now let's assume the system is expanded to 256 GB RAM, without reimaging the server - that is, still 32 GB Swap Space.

Based on the *Allocation Limit Formula*, Linux will not allocate all the memory:

```
Memory Allocation Limit = Swap Space + RAM * (Overcommit Ratio / 100)
Memory Allocation Limit = 32 GB + 256 GB * (50% / 100)
Memory Allocation Limit = 131 GB
```

Ooops! The system was expanded with 256 GB RAM, but is only using half of it. That is clearly not good.

```
Overcommit Ratio = 100 * ((RAM - Swap Space) / RAM)
Overcommit Ratio = 100 * ((256 GB - 32 GB) / 256 GB)
Overcommit Ratio = 87.5
```

The new setting for */proc/sys/vm/overcommit_ratio* must be *87.5* in order to utilize all available RAM, but not use the Swap.

Tweet (https://twitter.com/share)

© Copyright 2018 Pivotal Software, Inc. All Rights Reserved.
Privacy and Cookie Policy (https://pivotal.io/privacy-policy) • Terms of Use (https://pivotal.io/terms-of-use) • Product Guide (https://pivotal.io/product-guide) • Contribute (https://github.com/pivotal-legacy/blog/) •