Documentation for /proc/sys/vm/

kernel version 2.6.29

This file contains the documentation for the sysctl files in /proc/sys/vm and is valid for Linux kernel version 2.6.29.

The files in this directory can be used to tune the operation of the virtual memory (VM) subsystem of the Linux kernel and the writeout of dirty data to disk.

Default values and initialization routines for most of these files can be found in mm/swap.c.

Currently, these files are in /proc/sys/vm:

- admin_reserve_kbytes
- compact_memory
- compaction_proactiveness
- compact_unevictable_allowed
- dirty_background_bytes
- dirty_background_ratio
- dirty_bytes
- dirty_expire_centisecs
- dirty_ratio
- dirtytime_expire_seconds
- dirty_writeback_centisecs
- drop_caches
- extfrag_threshold
- highmem_is_dirtyable
- hugetlb_shm_group
- laptop_mode
- legacy_va_layout
- lowmem_reserve_ratio
- max_map_count
- memory_failure_early_kill
- memory_failure_recovery
- min_free_kbytes
- min_slab_ratio
- min_unmapped_ratio
- mmap_min_addr
- mmap_rnd_bits
- mmap_rnd_compat_bits
- nr_hugepages
- nr_hugepages_mempolicy
- nr_overcommit_hugepages
- nr_trim_pages (only if CONFIG_MMU=n)
- numa_zonelist_order
- oom_dump_tasks
- oom_kill_allocating_task
- overcommit_kbytes
- overcommit_memory
- overcommit_ratio
- page-cluster
- panic_on_oom
- percpu_pagelist_high_fraction
- stat_interval
- stat_refresh
- numa_stat
- swappiness
- unprivileged_userfaultfd
- user_reserve_kbytes
- vfs_cache_pressure
- watermark_boost_factor

- watermark_scale_factor
- zone_reclaim_mode

admin_reserve_kbytes

The amount of free memory in the system that should be reserved for users with the capability cap_sys_admin.

admin_reserve_kbytes defaults to min(3% of free pages, 8MB)

That should provide enough for the admin to log in and kill a process, if necessary, under the default overcommit 'guess' mode.

Systems running under overcommit 'never' should increase this to account for the full Virtual Memory Size of programs used to recover. Otherwise, root may not be able to log in to recover the system.

How do you calculate a minimum useful reserve?

sshd or login + bash (or some other shell) + top (or ps, kill, etc.)

For overcommit 'guess', we can sum resident set sizes (RSS). On x86_64 this is about 8MB.

For overcommit 'never', we can take the max of their virtual sizes (VSZ) and add the sum of their RSS. On x86_64 this is about 128MB.

Changing this takes effect whenever an application requests memory.

compact_memory

Available only when CONFIG_COMPACTION is set. When 1 is written to the file, all zones are compacted such that free memory is available in contiguous blocks where possible. This can be important for example in the allocation of huge pages although processes will also directly compact memory as required.

compaction_proactiveness

This tunable takes a value in the range [0, 100] with a default value of 20. This tunable determines how aggressively compaction is done in the background. Write of a non zero value to this tunable will immediately trigger the proactive compaction. Setting it to 0 disables proactive compaction.

Note that compaction has a non-trivial system-wide impact as pages belonging to different processes are moved around, which could also lead to latency spikes in unsuspecting applications. The kernel employs various heuristics to avoid wasting CPU cycles if it detects that proactive compaction is not being effective.

Be careful when setting it to extreme values like 100, as that may cause excessive background compaction activity.

compact_unevictable_allowed

Available only when CONFIG_COMPACTION is set. When set to 1, compaction is allowed to examine the unevictable lru (mlocked pages) for pages to compact. This should be used on systems where stalls for minor page faults are an acceptable trade for large contiguous free memory. Set to 0 to prevent compaction from moving pages that are unevictable. Default value is 1. On CONFIG_PREEMPT_RT the default value is 0 in order to avoid a page fault, due to compaction, which would block the task from becoming active until the fault is resolved.

dirty_background_bytes

Contains the amount of dirty memory at which the background kernel flusher threads will start writeback.

Note:

> dirty_background_bytes is the counterpart of dirty_background_ratio. Only one of them may be specified at a time. When one sysctl is written it is immediately taken into account to evaluate the dirty memory limits and the other appears as 0 when read.

dirty_background_ratio

Contains, as a percentage of total available memory that contains free pages and reclaimable pages, the number of pages at which the background kernel flusher threads will start writing out dirty data.

The total available memory is not equal to total system memory.

dirty_bytes

Contains the amount of dirty memory at which a process generating disk writes will itself start writeback.

Note: dirty_bytes is the counterpart of dirty_ratio. Only one of them may be specified at a time. When one sysctl is written it is

immediately taken into account to evaluate the dirty memory limits and the other appears as 0 when read.

Note: the minimum value allowed for dirty_bytes is two pages (in bytes); any value lower than this limit will be ignored and the old configuration will be retained.

dirty_expire_centisecs

This tunable is used to define when dirty data is old enough to be eligible for writeout by the kernel flusher threads. It is expressed in 100'ths of a second. Data which has been dirty in-memory for longer than this interval will be written out next time a flusher thread wakes up.

dirty_ratio

Contains, as a percentage of total available memory that contains free pages and reclaimable pages, the number of pages at which a process which is generating disk writes will itself start writing out dirty data.

The total available memory is not equal to total system memory.

dirtytime_expire_seconds

When a lazytime inode is constantly having its pages dirtied, the inode with an updated timestamp will never get chance to be written out. And, if the only thing that has happened on the file system is a dirtytime inode caused by an atime update, a worker will be scheduled to make sure that inode eventually gets pushed out to disk. This tunable is used to define when dirty inode is old enough to be eligible for writeback by the kernel flusher threads. And, it is also used as the interval to wakeup dirtytime_writeback thread.

dirty_writeback_centisecs

The kernel flusher threads will periodically wake up and write *old* data out to disk. This tunable expresses the interval between those wakeups, in 100'ths of a second.

Setting this to zero disables periodic writeback altogether.

drop_caches

Writing to this will cause the kernel to drop clean caches, as well as reclaimable slab objects like dentries and inodes. Once dropped, their memory becomes free.

To free pagecache:

```
echo 1 > /proc/sys/vm/drop_caches
```

To free reclaimable slab objects (includes dentries and inodes):

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

To free slab objects and pagecache:

```
echo 3 > /proc/sys/vm/drop_caches
```

This is a non-destructive operation and will not free any dirty objects. To increase the number of objects freed by this operation, the user may run *sync* prior to writing to /proc/sys/vm/drop_caches. This will minimize the number of dirty objects on the system and create more candidates to be dropped.

This file is not a means to control the growth of the various kernel caches (inodes, dentries, pagecache, etc...) These objects are automatically reclaimed by the kernel when memory is needed elsewhere on the system.

Use of this file can cause performance problems. Since it discards cached objects, it may cost a significant amount of I/O and CPU to recreate the dropped objects, especially if they were under heavy use. Because of this, use outside of a testing or debugging environment is not recommended.

You may see informational messages in your kernel log when this file is used:

```
cat (1234): drop_caches: 3
```

These are informational only. They do not mean that anything is wrong with your system. To disable them, echo 4 (bit 2) into drop_caches.

extfrag_threshold

This parameter affects whether the kernel will compact memory or direct reclaim to satisfy a high-order allocation. The extfrag/extfrag_index file in debugfs shows what the fragmentation index for each order is in each zone in the system. Values tending towards 0 imply allocations would fail due to lack of memory, values towards 1000 imply failures are due to fragmentation and -1

implies that the allocation will succeed as long as watermarks are met.

The kernel will not compact memory in a zone if the fragmentation index is <= extfrag_threshold. The default value is 500.

highmem_is_dirtyable

Available only for systems with CONFIG_HIGHMEM enabled (32b systems).

This parameter controls whether the high memory is considered for dirty writers throttling. This is not the case by default which means that only the amount of memory directly visible/usable by the kernel can be dirtied. As a result, on systems with a large amount of memory and lowmem basically depleted writers might be throttled too early and streaming writes can get very slow.

Changing the value to non zero would allow more memory to be dirtied and thus allow writers to write more data which can be flushed to the storage more effectively. Note this also comes with a risk of pre-mature OOM killer because some writers (e.g. direct block device writes) can only use the low memory and they can fill it up with dirty data without any throttling.

hugetlb_shm_group

hugetlb_shm_group contains group id that is allowed to create SysV shared memory segment using hugetlb page.

laptop_mode

laptop_mode is a knob that controls "laptop mode". All the things that are controlled by this knob are discussed in Documentation/admin-guide/laptops/laptop-mode.rst.

legacy_va_layout

If non-zero, this sysctl disables the new 32-bit mmap layout - the kernel will use the legacy (2.4) layout for all processes.

lowmem_reserve_ratio

For some specialised workloads on highmem machines it is dangerous for the kernel to allow process memory to be allocated from the "lowmem" zone. This is because that memory could then be pinned via the mlock() system call, or by unavailability of swapspace.

And on large highmem machines this lack of reclaimable lowmem memory can be fatal.

So the Linux page allocator has a mechanism which prevents allocations which *could* use highmem from using too much lowmem. This means that a certain amount of lowmem is defended from the possibility of being captured into pinned user memory.

(The same argument applies to the old 16 megabyte ISA DMA region. This mechanism will also defend that region from allocations which could use highmem or lowmem).

The *lowmem_reserve_ratio* tunable determines how aggressive the kernel is in defending these lower zones.

If you have a machine which uses highmem or ISA DMA and your applications are using mlock(), or if you are running with no swap then you probably should change the lowmem_reserve_ratio setting.

The lowmem_reserve_ratio is an array. You can see them by reading this file:

```
% cat /proc/sys/vm/lowmem_reserve_ratio
256     256     32
```

But, these values are not used directly. The kernel calculates # of protection pages for each zones from them. These are shown as array of protection pages in /proc/zoneinfo like followings. (This is an example of x86-64 box). Each zone has an array of protection pages like this:

```
Node 0, zone      DMA
  pages free     1355
        min      3
        low      3
        high     4
      :
      :
    numa_other   0
        protection: (0, 2004, 2004, 2004)
      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  pagesets
    cpu: 0 pcp: 0
        :
```

These protections are added to score to judge whether this zone should be used for page allocation or should be reclaimed.

In this example, if normal pages (index=2) are required to this DMA zone and watermark[WMARK_HIGH] is used for watermark, the kernel judges this zone should not be used because pages_free(1355) is smaller than watermark + protection[2] (4 + 2004 = 2008). If this protection value is 0, this zone would be used for normal page requirement. If requirement is DMA zone(index=0),

protection[0] (=0) is used.

zone[i]'s protection[j] is calculated by following expression:

```
(i < j):
  zone[i]->protection[j]
  = (total sums of managed_pages from zone[i+1] to zone[j] on the node)
    / lowmem_reserve_ratio[i];
(i = j):
   (should not be protected. = 0;
(i > j):
   (not necessary, but looks 0)
```

The default values of lowmem_reserve_ratio[i] are

256	(if zone[i] means DMA or DMA32 zone)
32	(others)

As above expression, they are reciprocal number of ratio. 256 means 1/256. # of protection pages becomes about "0.39%" of total managed pages of higher zones on the node.

If you would like to protect more pages, smaller values are effective. The minimum value is 1 (1/1 -> 100%). The value less than 1 completely disables protection of the pages.

max_map_count:

This file contains the maximum number of memory map areas a process may have. Memory map areas are used as a side-effect of calling malloc, directly by mmap, mprotect, and madvise, and also when loading shared libraries.

While most applications need less than a thousand maps, certain programs, particularly malloc debuggers, may consume lots of them, e.g., up to one or two maps per allocation.

The default value is 65530.

memory_failure_early_kill:

Control how to kill processes when uncorrected memory error (typically a 2bit error in a memory module) is detected in the background by hardware that cannot be handled by the kernel. In some cases (like the page still having a valid copy on disk) the kernel will handle the failure transparently without affecting any applications. But if there is no other uptodate copy of the data it will kill to prevent any data corruptions from propagating.

1: Kill all processes that have the corrupted and not reloadable page mapped as soon as the corruption is detected. Note this is not supported for a few types of pages, like kernel internally allocated data or the swap cache, but works for the majority of user pages.

0: Only unmap the corrupted page from all processes and only kill a process who tries to access it.

The kill is done using a catchable SIGBUS with BUS_MCEERR_AO, so processes can handle this if they want to.

This is only active on architectures/platforms with advanced machine check handling and depends on the hardware capabilities.

Applications can override this setting individually with the PR_MCE_KILL prctl

memory_failure_recovery

Enable memory failure recovery (when supported by the platform)

1: Attempt recovery.

0: Always panic on a memory failure.

min_free_kbytes

This is used to force the Linux VM to keep a minimum number of kilobytes free. The VM uses this number to compute a watermark[WMARK_MIN] value for each lowmem zone in the system. Each lowmem zone gets a number of reserved free pages based proportionally on its size.

Some minimal amount of memory is needed to satisfy PF_MEMALLOC allocations; if you set this to lower than 1024KB, your system will become subtly broken, and prone to deadlock under high loads.

Setting this too high will OOM your machine instantly.

min_slab_ratio

This is available only on NUMA kernels.

A percentage of the total pages in each zone. On Zone reclaim (fallback from the local zone occurs) slabs will be reclaimed if more than this percentage of pages in a zone are reclaimable slab pages. This insures that the slab growth stays under control even in NUMA systems that rarely perform global reclaim.

The default is 5 percent.

Note that slab reclaim is triggered in a per zone / node fashion. The process of reclaiming slab memory is currently not node specific and may not be fast.

min_unmapped_ratio

This is available only on NUMA kernels.

This is a percentage of the total pages in each zone. Zone reclaim will only occur if more than this percentage of pages are in a state that zone_reclaim_mode allows to be reclaimed.

If zone_reclaim_mode has the value 4 OR'd, then the percentage is compared against all file-backed unmapped pages including swapcache pages and tmpfs files. Otherwise, only unmapped pages backed by normal files but not tmpfs files and similar are considered.

The default is 1 percent.

mmap_min_addr

This file indicates the amount of address space which a user process will be restricted from mmapping. Since kernel null dereference bugs could accidentally operate based on the information in the first couple of pages of memory userspace processes should not be allowed to write to them. By default this value is set to 0 and no protections will be enforced by the security module. Setting this value to something like 64k will allow the vast majority of applications to work correctly and provide defense in depth against future potential kernel bugs.

mmap_rnd_bits

This value can be used to select the number of bits to use to determine the random offset to the base address of vma regions resulting from mmap allocations on architectures which support tuning address space randomization. This value will be bounded by the architecture's minimum and maximum supported values.

This value can be changed after boot using the /proc/sys/vm/mmap_rnd_bits tunable

mmap_rnd_compat_bits

This value can be used to select the number of bits to use to determine the random offset to the base address of vma regions resulting from mmap allocations for applications run in compatibility mode on architectures which support tuning address space randomization. This value will be bounded by the architecture's minimum and maximum supported values.

This value can be changed after boot using the /proc/sys/vm/mmap_rnd_compat_bits tunable

nr_hugepages

Change the minimum size of the hugepage pool.

See Documentation/admin-guide/mm/hugetlbpage.rst

nr_hugepages_mempolicy

Change the size of the hugepage pool at run-time on a specific set of NUMA nodes.

See Documentation/admin-guide/mm/hugetlbpage.rst

nr_overcommit_hugepages

Change the maximum size of the hugepage pool. The maximum is nr_hugepages + nr_overcommit_hugepages.

See Documentation/admin-guide/mm/hugetlbpage.rst

nr_trim_pages

This is available only on NOMMU kernels.

This value adjusts the excess page trimming behaviour of power-of-2 aligned NOMMU mmap allocations.

A value of 0 disables trimming of allocations entirely, while a value of 1 trims excess pages aggressively. Any value >= 1 acts as the

watermark where trimming of allocations is initiated.

The default value is 1.

See Documentation/admin-guide/mm/nommu-mmap.rst for more information.

numa_zonelist_order

This sysctl is only for NUMA and it is deprecated. Anything but Node order will fail!

'where the memory is allocated from' is controlled by zonelists.

(This documentation ignores ZONE_HIGHMEM/ZONE_DMA32 for simple explanation. you may be able to read ZONE_DMA as ZONE_DMA32...)

In non-NUMA case, a zonelist for GFP_KERNEL is ordered as following. ZONE_NORMAL -> ZONE_DMA This means that a memory allocation request for GFP_KERNEL will get memory from ZONE_DMA only when ZONE_NORMAL is not available.

In NUMA case, you can think of following 2 types of order. Assume 2 node NUMA and below is zonelist of Node(0)'s GFP_KERNEL:

```
(A) Node(0) ZONE_NORMAL -> Node(0) ZONE_DMA -> Node(1) ZONE_NORMAL
(B) Node(0) ZONE_NORMAL -> Node(1) ZONE_NORMAL -> Node(0) ZONE_DMA.
```

Type(A) offers the best locality for processes on Node(0), but ZONE_DMA will be used before ZONE_NORMAL exhaustion. This increases possibility of out-of-memory(OOM) of ZONE_DMA because ZONE_DMA is tend to be small.

Type(B) cannot offer the best locality but is more robust against OOM of the DMA zone.

Type(A) is called as "Node" order. Type (B) is "Zone" order.

"Node order" orders the zonelists by node, then by zone within each node. Specify "[Nn]ode" for node order

"Zone Order" orders the zonelists by zone type, then by node within each zone. Specify "[Zz]one" for zone order.

Specify "[Dd]efault" to request automatic configuration.

On 32-bit, the Normal zone needs to be preserved for allocations accessible by the kernel, so "zone" order will be selected.

On 64-bit, devices that require DMA32/DMA are relatively rare, so "node" order will be selected.

Default order is recommended unless this is causing problems for your system/application.

oom_dump_tasks

Enables a system-wide task dump (excluding kernel threads) to be produced when the kernel performs an OOM-killing and includes such information as pid, uid, tgid, vm size, rss, pgtables_bytes, swapents, oom_score_adj score, and name. This is helpful to determine why the OOM killer was invoked, to identify the rogue task that caused it, and to determine why the OOM killer chose the task it did to kill.

If this is set to zero, this information is suppressed. On very large systems with thousands of tasks it may not be feasible to dump the memory state information for each one. Such systems should not be forced to incur a performance penalty in OOM conditions when the information may not be desired.

If this is set to non-zero, this information is shown whenever the OOM killer actually kills a memory-hogging task.

The default value is 1 (enabled).

oom_kill_allocating_task

This enables or disables killing the OOM-triggering task in out-of-memory situations.

If this is set to zero, the OOM killer will scan through the entire tasklist and select a task based on heuristics to kill. This normally selects a rogue memory-hogging task that frees up a large amount of memory when killed.

If this is set to non-zero, the OOM killer simply kills the task that triggered the out-of-memory condition. This avoids the expensive tasklist scan.

If panic_on_oom is selected, it takes precedence over whatever value is used in oom_kill_allocating_task.

The default value is 0.

overcommit_kbytes

When overcommit_memory is set to 2, the committed address space is not permitted to exceed swap plus this amount of physical RAM. See below.

Note: overcommit_kbytes is the counterpart of overcommit_ratio. Only one of them may be specified at a time. Setting one disables the other (which then appears as 0 when read).

overcommit_memory

This value contains a flag that enables memory overcommitment.

When this flag is 0, the kernel attempts to estimate the amount of free memory left when userspace requests more memory.

When this flag is 1, the kernel pretends there is always enough memory until it actually runs out.

When this flag is 2, the kernel uses a "never overcommit" policy that attempts to prevent any overcommit of memory. Note that user_reserve_kbytes affects this policy.

This feature can be very useful because there are a lot of programs that malloc() huge amounts of memory "just-in-case" and don't use much of it.

The default value is 0.

See Documentation/vm/overcommit-accounting.rst and mm/util.c::__vm_enough_memory() for more information.

overcommit_ratio

When overcommit_memory is set to 2, the committed address space is not permitted to exceed swap plus this percentage of physical RAM. See above.

page-cluster

page-cluster controls the number of pages up to which consecutive pages are read in from swap in a single attempt. This is the swap counterpart to page cache readahead. The mentioned consecutivity is not in terms of virtual/physical addresses, but consecutive on swap space - that means they were swapped out together.

It is a logarithmic value - setting it to zero means "1 page", setting it to 1 means "2 pages", setting it to 2 means "4 pages", etc. Zero disables swap readahead completely.

The default value is three (eight pages at a time). There may be some small benefits in tuning this to a different value if your workload is swap-intensive.

Lower values mean lower latencies for initial faults, but at the same time extra faults and I/O delays for following faults if they would have been part of that consecutive pages readahead would have brought in.

panic_on_oom

This enables or disables panic on out-of-memory feature.

If this is set to 0, the kernel will kill some rogue process, called oom_killer. Usually, oom_killer can kill rogue processes and system will survive.

If this is set to 1, the kernel panics when out-of-memory happens. However, if a process limits using nodes by mempolicy/cpusets, and those nodes become memory exhaustion status, one process may be killed by oom-killer. No panic occurs in this case. Because other nodes' memory may be free. This means system total status may be not fatal yet.

If this is set to 2, the kernel panics compulsorily even on the above-mentioned. Even oom happens under memory cgroup, the whole system panics.

The default value is 0.

1 and 2 are for failover of clustering. Please select either according to your policy of failover.

panic_on_oom=2+kdump gives you very strong tool to investigate why oom happens. You can get snapshot.

percpu_pagelist_high_fraction

This is the fraction of pages in each zone that are can be stored to per-cpu page lists. It is an upper boundary that is divided depending on the number of online CPUs. The min value for this is 8 which means that we do not allow more than 1/8th of pages in each zone to be stored on per-cpu page lists. This entry only changes the value of hot per-cpu page lists. A user can specify a number like 100 to allocate 1/100th of each zone between per-cpu lists.

The batch value of each per-cpu page list remains the same regardless of the value of the high fraction so allocation latencies are unaffected.

The initial value is zero. Kernel uses this value to set the high pcp->high mark based on the low watermark for the zone and the number of local online CPUs. If the user writes '0' to this sysctl, it will revert to this default behavior.

stat_interval

The time interval between which vm statistics are updated. The default is 1 second.

stat_refresh

Any read or write (by root only) flushes all the per-cpu vm statistics into their global totals, for more accurate reports when testing e.g. cat /proc/sys/vm/stat_refresh /proc/meminfo

As a side-effect, it also checks for negative totals (elsewhere reported as 0) and "fails" with EINVAL if any are found, with a warning in dmesg. (At time of writing, a few stats are known sometimes to be found negative, with no ill effects: errors and warnings on these stats are suppressed.)

numa_stat

This interface allows runtime configuration of numa statistics.

When page allocation performance becomes a bottleneck and you can tolerate some possible tool breakage and decreased numa counter precision, you can do:

```
echo 0 > /proc/sys/vm/numa_stat
```

When page allocation performance is not a bottleneck and you want all tooling to work, you can do:

```
echo 1 > /proc/sys/vm/numa_stat
```

swappiness

This control is used to define the rough relative IO cost of swapping and filesystem paging, as a value between 0 and 200. At 100, the VM assumes equal IO cost and will thus apply memory pressure to the page cache and swap-backed pages equally; lower values signify more expensive swap IO, higher values indicates cheaper.

Keep in mind that filesystem IO patterns under memory pressure tend to be more efficient than swap's random IO. An optimal value will require experimentation and will also be workload-dependent.

The default value is 60.

For in-memory swap, like zram or zswap, as well as hybrid setups that have swap on faster devices than the filesystem, values beyond 100 can be considered. For example, if the random IO against the swap device is on average 2x faster than IO from the filesystem, swappiness should be 133 (x + 2x = 200, 2x = 133.33).

At 0, the kernel will not initiate swap until the amount of free and file-backed pages is less than the high watermark in a zone.

unprivileged_userfaultfd

This flag controls the mode in which unprivileged users can use the userfaultfd system calls. Set this to 0 to restrict unprivileged users to handle page faults in user mode only. In this case, users without SYS_CAP_PTRACE must pass UFFD_USER_MODE_ONLY in order for userfaultfd to succeed. Prohibiting use of userfaultfd for handling faults from kernel mode may make certain vulnerabilities more difficult to exploit.

Set this to 1 to allow unprivileged users to use the userfaultfd system calls without any restrictions.

The default value is 0.

user_reserve_kbytes

When overcommit_memory is set to 2, "never overcommit" mode, reserve min(3% of current process size, user_reserve_kbytes) of free memory. This is intended to prevent a user from starting a single memory hogging process, such that they cannot recover (kill the hog).

user_reserve_kbytes defaults to min(3% of the current process size, 128MB).

If this is reduced to zero, then the user will be allowed to allocate all free memory with a single process, minus admin_reserve_kbytes. Any subsequent attempts to execute a command will result in "fork: Cannot allocate memory".

Changing this takes effect whenever an application requests memory.

vfs_cache_pressure

This percentage value controls the tendency of the kernel to reclaim the memory which is used for caching of directory and inode objects.

At the default value of vfs_cache_pressure=100 the kernel will attempt to reclaim dentries and inodes at a "fair" rate with respect to pagecache and swapcache reclaim. Decreasing vfs_cache_pressure causes the kernel to prefer to retain dentry and inode caches. When vfs_cache_pressure=0, the kernel will never reclaim dentries and inodes due to memory pressure and this can easily lead to out-of-memory conditions. Increasing vfs_cache_pressure beyond 100 causes the kernel to prefer to reclaim dentries and inodes.

Increasing vfs_cache_pressure significantly beyond 100 may have negative performance impact. Reclaim code needs to take various locks to find freeable directory and inode objects. With vfs_cache_pressure=1000, it will look for ten times more freeable objects than there are.

watermark_boost_factor

This factor controls the level of reclaim when memory is being fragmented. It defines the percentage of the high watermark of a zone that will be reclaimed if pages of different mobility are being mixed within pageblocks. The intent is that compaction has less work to do in the future and to increase the success rate of future high-order allocations such as SLUB allocations, THP and hugetlbfs pages.

To make it sensible with respect to the watermark_scale_factor parameter, the unit is in fractions of 10,000. The default value of 15,000 means that up to 150% of the high watermark will be reclaimed in the event of a pageblock being mixed due to fragmentation. The level of reclaim is determined by the number of fragmentation events that occurred in the recent past. If this value is smaller than a pageblock then a pageblocks worth of pages will be reclaimed (e.g. 2MB on 64-bit x86). A boost factor of 0 will disable the feature.

watermark_scale_factor

This factor controls the aggressiveness of kswapd. It defines the amount of memory left in a node/system before kswapd is woken up and how much memory needs to be free before kswapd goes back to sleep.

The unit is in fractions of 10,000. The default value of 10 means the distances between watermarks are 0.1% of the available memory in the node/system. The maximum value is 3000, or 30% of memory.

A high rate of threads entering direct reclaim (allocstall) or kswapd going to sleep prematurely (kswapd_low_wmark_hit_quickly) can indicate that the number of free pages kswapd maintains for latency reasons is too small for the allocation bursts occurring in the system. This knob can then be used to tune kswapd aggressiveness accordingly.

zone_reclaim_mode

Zone_reclaim_mode allows someone to set more or less aggressive approaches to reclaim memory when a zone runs out of memory. If it is set to zero then no zone reclaim occurs. Allocations will be satisfied from other zones / nodes in the system.

This is value OR'ed together of

1	Zone reclaim on
2	Zone reclaim writes dirty pages out
4	Zone reclaim swaps pages

zone_reclaim_mode is disabled by default. For file servers or workloads that benefit from having their data cached, zone_reclaim_mode should be left disabled as the caching effect is likely to be more important than data locality.

Consider enabling one or more zone_reclaim mode bits if it's known that the workload is partitioned such that each partition fits within a NUMA node and that accessing remote memory would cause a measurable performance reduction. The page allocator will take additional actions before allocating off node pages.

Allowing zone reclaim to write out pages stops processes that are writing large amounts of data from dirtying pages on other nodes. Zone reclaim will write out dirty pages if a zone fills up and so effectively throttle the process. This may decrease the performance of a single process since it cannot use all of system memory to buffer the outgoing writes anymore but it preserve the memory on other nodes so that the performance of other processes running on other nodes will not be affected.

Allowing regular swap effectively restricts allocations to the local node unless explicitly overridden by memory policies or cpuset configurations.