# ⛳ All in One, Tips & Tricks for tuning & Maintaining PostgreSQL for DBAs⛳

## Part B:

## Query Cost Configuration Parameters

The query planner uses a variety of configuration parameters and signals to calculate the cost of every query. Some of these parameters are listed below and can potentially improve the performance of a PostgreSQL query.

## --random_page_cost

sets the cost for the planner for fetching a disk page non-sequentially.
The default value is 4.
You can configure this at the tablespace level as well.
Reducing the value to less than 4 will make the planner prefer index scans.
Increasing it will tell the planner that index scans are more expensive.
So on a pure OLTO environment decrease this can be better.

Random access to mechanical disk storage is normally much more expensive than four times sequential access. However, a lower default is used (4.0) because the majority of random accesses to disk, such as indexed reads, are assumed to be in cache.
The default value can be thought of as modeling random access as 40 times slower than sequential, while expecting 90% of random reads to be cached.
If you believe a 90% cache rate is an incorrect assumption for your workload, you can increase random_page_cost to better reflect the true cost of random storage reads. Correspondingly, if your data is likely to be completely in cache, such as when the database is smaller than the total server memory, decreasing random_page_cost can be appropriate.
Storage that has a low random read cost relative to sequential, e.g., solid-state drives(SSD), might also be better modeled with a lower value for random_page_cost, e.g., 1.1.

### random_page_cost bs than seq_page_cos?

Although the system will let you set random_page_cost to less than seq_page_cost, it is not physically sensible to do so.
However, setting them equal makes sense if the database is entirely cached in RAM, since in that case there is no penalty for touching pages out of sequence.
Also, in a heavily-cached database you should lower both values relative to the CPU parameters, since the cost of fetching a page already in RAM is much smaller than it would normally be.

## - Effective_cache_size

Sets the planner's assumption about the effective size of the disk cache that is available to a single query.
This is factored into estimates of the cost of using an index; a higher value makes it more likely index scans will be used, a lower value makes it more likely sequential scans will be used.

When setting this parameter you should consider both PostgreSQL's shared buffers and the portion of the kernel's disk cache that will be used for PostgreSQLdata files, though some data might exist in both places.

Also, take into account the expected number of concurrent queries on different tables, since they will have to share the available space. This parameter has no effect on the size of shared memory allocated by PostgreSQL, nor does it reserve kernel disk cache; it is used only for estimation purposes. The system also does not assume data remains in the disk cache between queries. If this value is specified without units, it is taken as blocks, that is BLCKSZ bytes, typically 8kB. The default is 4 gigabytes (4GB). (If BLCKSZ is not 8kB, the default value scales proportionally to it.)

**RECOMMENDATIONS**
Tells the PostgreSQL query planner how much RAM is estimated to be available for caching data, in both shared_buffers and in the filesystem cache. This setting just helps the planner make good cost estimates; it does not actually allocate the memory.
Recommendations are to set Effective_cache_size at 50% of the machine's total RAM or 3/4 total RAM.
Total RAM means Total RAM really consider for Postgres, so all needed memory for other applications on your server was decreased
from it.

So if you have a large index in your database, yoh need increase this parameter if optimizer have problems on your index selection logically.
Also consider decreasing **random_page_cost.**

**Note**: we can consider Effective_cache_size parameter equal to OPTIMIZER_INDEX_COST_ADJ in Oracle.

Oracle: OPTIMIZER_INDEX_COST_ADJ lets you tune optimizer behavior for access path selection to be more or less index friendly, that is, to make the optimizer more or less prone to selecting an index access path over a full table scan.

**Using a preferred usage of Effective cache can be depends on Linux settings about cache configuration.**

# Overview of OS buffer cache:

The buffer cache is a memory region that Linux uses to make read operations faster. We'll first go over the basics of the buffer cache and the reasons why we need it. Next, we'll go over how to clear it, to reclaim the occupied memory. Finally, we'll show how to restrict its size.

# Tuning Page Cache

File system caching in Linux is a mechanism that allows the kernel to store frequently accessed data in memory for faster access. **The kernel uses the page cache to store recently-read data from files and file system metadata**.
For instance, when a program reads data from a file, the kernel performs several tasks:
- checks the page cache to see if the data is already in memory
- if the data is in memory, the kernel simply returns the data from the cache
- otherwise, it reads the data from the drive and stores a copy of it in the cache for future use.

In addition, **the kernel uses the dentries cache to store information about file system objects**. These file system objects include directories and inodes.

Hence, **the page cache handles file system metadata while the dentries cache manages the file system objects**.
Again, **the kernel uses a Least Recently Used (LRU) algorithm to manage the page and dentries cache.**
In other words, when the cache is full and there's more data to add, the kernel removes the least recently used data to make room for the new data.

Page Cache is a disk cache which holds data of files and executable programs, e.g. pages with actual contents of files or block devices. Page Cache (disk cache) is used to reduce the number of disk reads.

The Page Cache in Red Hat Enterprise Linux  is dynamically adjusted. You can adjust the minimum free pages using

# echo 1024 > /proc/sys/vm/min_free_kbytes

Again to make the change permanent, add the following line to the file /etc/sysctl.conf:
# echo vm.min_free_kbytes=1024 >> /etc/sysctl.conf

The easiest way to tune when to start reclaiming Page Cache pages is to adjust the swappiness percentage.

The default value of /proc/sys/vm/swappiness is 60 which means that applications that are not actively occupying CPU cycles can be swapped out.
Higher values will provide more I/O cache and lower values will wait longer to swap out idle applications.

Ensure the following settings are present in /etc/sysctl.conf and set by your DBA to preferred values.

**vm.swappiness=1**
**vm.dirty_background_ratio=3**
**vm.dirty_ratio=15**
**vm.dirty_expire_centisecs=500**
**vm.dirty_writeback_centisecs=100**

# Checking the Cache

**The *vmstat* command provides detailed information about virtual memory**. In particular, it shows the amount of memory in use for caching:

```
$ vmstat
procs  -memory--swap-io--system--cpu-----
r   b  swpd   free   buff  cache  si  so  bi  bo  in    cs    us sy id wa st
0   0     0  6130448 11032 589532    0   0   422    52     160   362    3  3 76 18  0
```

**The *cache* column shows the amount of memory used for file system caching in kilobytes**.
In addition, to get more details using the *vmstat* command, we can use the *-s* flag:

```
$vmstat -s
8016140 K total memory
1282340 K used memory
 207744 K active memory
 711356 K inactive memory
 6133536 K free memory
 11032 K buffer memory
 589232 K swap cache
 2097148 K total swap
 0 K used swap
 2097148 K free swap
```

3458 non-nice user cpu ticks
389 nice user cpu ticks
3371 system cpu ticks
60823 idle cpu ticks
20782 IO-wait cpu ticks
0 IRQ cpu ticks
34 softirq cpu ticks
0 stolen cpu ticks
494275 pages paged in
56168 pages paged out
0 pages swapped in
0 pages swapped out
170063 interrupts
384058 CPU context switches
1673971944 boot time
5151 forks

Alternatively, we can use the *free* command to check the amount of file system cache memory in the system. It shows the memory usage in kilobytes under the *buff/cache* column:

$ free
total   used   free  shared  buff/cache   available
**Mem**:
8016140   1284652   6130952   144680   600536   6353032
**Swap**:   2097148   0   2097148

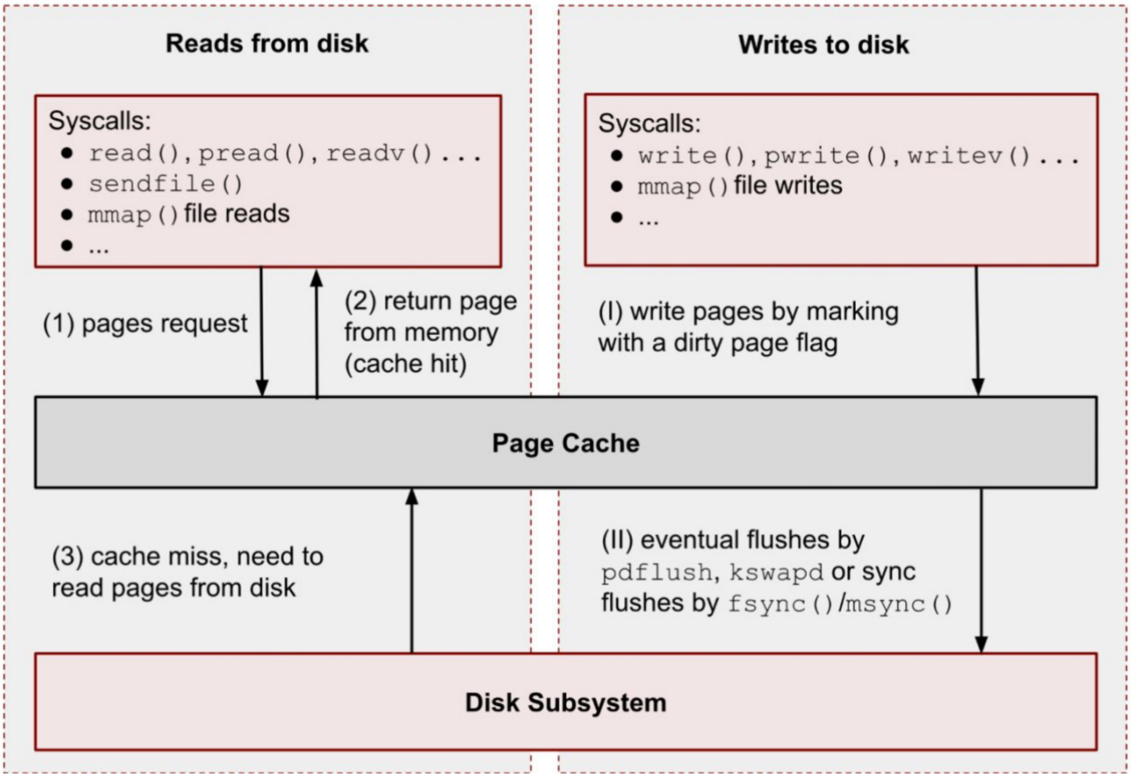
**The** *-m* flag alters the command output values to megabytes.
Notably, the value of the ***buff/cache*** column is the sum of the values of the ***buffer memory*** and ***swap cache*** rows for ***vmstat*****.**

## Next, let's configure the file system cache for our system.

# The OS Buffer Cache

At its core, **every computer program reads and writes data.** The speed at which data is transferred varies depending on the medium, and **it greatly affects the overall performance** and the response times experienced by users. For this reason, Linux tries to minimize the number of I/O operations with those buffer caches.
**Buffer caches live in memory**, thus reading and writing to them is much faster compared to disks. For this reason, instead of going to the disk every time, **Linux performs those write operations to those buffer caches**. It then does some bookkeeping so that it knows which piece of data is cached, and **it schedules regular flushes of that data onto the actual disks**.
While using buffer caches can greatly improve the overall performance, **they can sometimes claim a significant part of the main memory.** This can add additional latency when we want to reclaim that memory and is therefore **negatively impacting the running processes**. In the following sections, we'll examine how to mitigate that side-effect by manually clearing those caches and restricting the maximum size that they can occupy.

**OS Buffer cache is same value shown in Free command in buffer/cache column.**




# How to Clear the Buffer Cache

Imagine that we intend to launch a process that needs most of the available physical memory on our machine. Linux would have to flush the buffer caches onto the disk to reclaim this memory and this can be done by calling **fsync** command by PostgreSQL. This, however, takes time and thus can cause significant delays. To solve this issue, **we can proactively flush those caches.** To do that, we need to write to the drop_caches file in *procfs*. Let's see how to do this:

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

Note that the *drop_caches* file within *procfs* allows us to choose which type of buffer cache files we want to flush.
By this way only OS cache was clear and this not related on buffer cache of database.

# Configuring File System Cache

In general, **we can use the *sysctl* command to configure the file system cache in Linux**. Also, the *sysctl* command can modify kernel parameters in the */etc/sysctl.conf* file. This file contains system-wide kernel parameters that we can set at runtime.

# Setting *vfs_cache_pressure*

With *sysctl*, we can also set the value of ***vm.vfs_cache_pressure*, which controls the tendency of the kernel to reclaim the memory used for caching directory and inode objects**:

$ sysctl -w vm.vfs_cache_pressure=50

$vi /etc/sysctl.conf
vm.vfs_cache_pressure = 50

Here, we set the *vfs_cache_pressure* value to *50* via the *-w* switch of *sysctl*. Consequently, the kernel will prefer inode and dentry caches over the page cache. This can help improve performance on systems with a large number of files.

Notably, **a higher value makes the kernel prefer to reclaim inodes and dentries over cached memory and on databases this is not a good option** . On the other hand, a lower value makes it reclaim cached memory over inodes and entries. Hence, we can adjust the value according to our preference.
The default value of that *sysfs* option is 100. Setting this to a greater value, like 150 or 200, on systems running data-intensive applications has been known to help.
Next, let's set the value of swappiness.

# Configuring Swappiness

**Swappiness controls how aggressively the kernel swaps memory pages**. Lowering the value of swappiness means the kernel will be less likely to swap out less frequently used memory pages. Thus, the kernel will be more likely to keep these pages cached in RAM for faster access.
Further, we can again use *sysctl* to set the *vm.swappiness* parameter:

```
$ sudo sysctl -w vm.swappiness=10
vm.swappiness = 10
```

Here, the command sets the value of *vm.swappiness* to *10.* Again, **lower values will make the kernel prefer to keep more data in RAM** and this is optimal status for database environments because swapping has performance penalty. Thus, higher values make the kernel swap more.

For database environments we consider this to value 1 to avoid swapping, but you should optimize shared_buffer to logically doesn't need swapping.

Next, let's see other parameters we can set to configure the cache.

# File System Cache Optimization

To optimize file system caching, we can modify several parameters:

- *vm.dirty_background_ratio*
- *vm.dirty_background_bytes*
- *vm.dirty_ratio*
- *vm.dirty_bytes*
- *vm.dirty_writeback_centisecs*
- *vm.dirty_expire_centisecs*

**These parameters control the percentage of total system memory we can use for caching.** They regulate the caching memory before the kernel writes dirty pages to the storage. Importantly, **dirty pages are memory pages that aren't written to secondary memory yet**.
Let's see the *dirty_** variables on our system using the *sysctl* command:

```
$ sysctl -a | grep dirty
vm.dirty_background_ratio = 10
vm.dirty_background_bytes = 0
vm.dirty_ratio = 20
vm.dirty_bytes = 0
vm.dirty_expire_centisecs = 3000
vm.dirty_writeback_centisecs = 500
```

Here, the *-a* option displays all the variables we can set along with their values. Then the *grep* command filters all the *vm.dirty_**variables.
Let's start with a summary of how these parameters work.

## vm.dirty_background_ratio

The *vm.dirty_background_ratio* parameter is the amount of system memory in percentage that can be filled with dirty pages before they're written to the drive.
For instance, if we set the value of the *vm.dirty_background_ratio* parameter of a 64GB RAM system to 10, it entails that 6.4GB of data (dirty pages) can stay in RAM before they're written to the storage.
Now, let's configure the value of *vm.dirty_background_ratio* for our system:

```
$ sudo sysctl -w vm.dirty_background_ratio=10
 vm.dirty_background_ratio = 10
```

Alternatively, **we can set the *vm.dirty_background_bytes* variable in place of *vm.dirty_background_ratio*.** The *_bytes* version takes the amount of memory in bytes. For example, we can set the amount of memory for dirty background caching to 512MB:

```
$ sudo sysctl -w vm.dirty_background_bytes=511870912
```

However, **the *_ratio* variant will become *0* if we set the * _bytes variant, and vice versa**.
Next, let's look at *vm.dirty_ratio*.

## vm.dirty_ratio

Specifically, *vm.dirty_ratio* is the absolute maximum amount of system memory in percentage that can be filled with dirty pages before they're written to the drive. At this level, all new I/O activities halt until dirty pages are written to storage.
Notably, the *vm.dirty_bytes* turns to *0* when we set a value in bytes for *vm.dirty_ratio* and vice versa. To illustrate, let's define the value for *vm.dirty_ratio*:

```
$ sudo sysctl -w vm.dirty_ratio=20
 vm.dirty_ratio = 20
```

Similarly, the *vm.dirty_ratio* will become *0* if we configure a value for the *vm.dirty _bytes*.

## The *_centisecs Variables

Of course, data cached in the system memory is at risk of loss in case of a power outage. Hence, to safeguard the system from data loss, the following variables dictate how long and how often data is written to secondary storage:

- *vm.dirty_expire_centisecs*
- *vm.dirty_writeback_centisecs*

The *vm.dirty_expire_centisecs* manages how long data can be in the cache before it's written to drive.
Let's set the variable so that data can stay for 40 seconds in the cache:

```
$ sudo sysctl -w vm.dirty_expire_centisecs=4000
vm.dirty_expire_centisecs = 4000
```

In this case, cached info can stay up to 40 seconds before it's written to the drive. Notably, 1s equals 100 centisecs.
Further, the *vm.dirty_writeback_centisecs* is the variable for how often the write background process checks to see if there's data to write to secondary storage. Thus, the lower the value, the higher the frequency, and vice versa.

Let's configure *vm.dirty_writeback_centisecs* to check the cache every 5 seconds:

```
$ sudo sysctl -w vm.dirty_writeback_centisecs=500
vm.dirty_writeback_centisecs = 500
```

Again, **the 500 centisecs value is equal to 5 seconds.** Next, let's make our configurations permanent.

Now, let’s open */etc/sysctl.conf* in an editor and add the earlier configurations to it:
vm.vfs_cache_pressure=50
vm.swappiness=10
vm.dirty_ratio=20
vm.dirty_background_ratio=10
vm.dirty_expire_centisecs=4000
vm.dirty_writeback_centisecs=500

However, we can also use the *cat* command with the full path to the variable:
$ cat /proc/sys/vm/vfs_cache_pressure
10
Again, we maybe need to apply these commands to any parameter to confirm our settings as optimally.

Sincerely,
Alireza Kamrani
(Database Box)