# Tuning PostgreSQL

PostgreSQL jest jedną z najbardziej zaawansowanych baz na świecie, wydajnością dorównuje liderom, jednakże parametry standardowej instalacji raczej nie są zaprojektowane do osiągania największej wydajności.

W celu zwiększenia wydajności na większości współczesnych maszyn należy przeprowadzić zmianę niektórych parametrów. Spośród nich najważniejsze to:

## shared_buffers

Konfiguracja shared_buffers określa, ile pamięci jest poświęcone dla PostgreSQL do buforowania danych. W większości przypadków, shared_buffers optymalnie będzie ustawić na 1/4 pamięci w systemie. Należy pamiętać, że w systemie Windows, duże wartości shared_buffers nie są tak skuteczne, prawdopodobnie lepsze wyniki można otrzymać utrzymując stosunkowo niskie shared_buffers, pozwalając korzystać z pamięci podręcznej systemu operacyjnego.

(8GB - jeżeli w systemie posiadamy 8GB RAM). Więcej o tuningu Linux dla baz danych

Zanim ustawi się shared_buffers należy zwiększyć ustawienie w systemie operacyjnym *kernel.shmmax*. W tym celu należy dopisać do pliku:

```
root# vim /etc/sysctl.conf (stuknij i - aby edytować)
kernel.shmmax=8589934592
kernel.shmall=4194304
:wq (zapisz i wyjdź)

root#sysctl -p
```

Edycja postgresql.conf

```
vim /etc/postgresql/9.3/postgresql.conf
shared_buffers = 196MB
```

## effective_cache_size

Należy ustawić ile pamięci jest do buforowania dysku pozostałości po uwzględnieniu tego, co jest używane przez system operacyjny, dedykowanej pamięci PostgreSQL, i innych aplikacji. Jeśli jest zbyt niska, indeksy nie mogą być wykorzystywane do wykonywania kwerend w taki sposób jaki można się spodziewać. Ustawianie effective_cache_size do 1/2 całkowitej pamięci jest, z reguły, najbardziej optymalnym ustawieniem. Można lepiej oszacować to ustawienie patrząc na statystyki OS. W systemach uniksowych, dodać należy wartości *free* + *cached* z polecenia *free*.

```
[root@edokumenty ~]# free
             total       used       free     shared    buffers     cached
Mem:       1035236     948720      86516          0      11688     761988
-/+ buffers/cache:     175044     860192
Swap:       497972      19040     478932
```

```
/etc/postgresql/9.3/postgresql.conf
effective_cache_size = 1024MB
```

W systemie Windows sprawdzić to można Menedżerze zadań na zakładce Wydajność - pole *Buforowana*.

## work_mem

Work_mem ustawić należy na nieco wyższą wartość niż jest domyślnie, ale należy uważać aby nie przesadzić.

Pamięć ta jest używana głównie do sortowania. Nie jest to wartość maksymalna, zn. tyle ile ustawimy, tyle proces zawsze zarezerwuje. Jeśli ustawimy go na 32MB, i mamy 30 użytkowników to wkrótce obciążenie wzrośnie do ~1GB pamięci rzeczywistej. Jeśli tyle posiadamy w systemie - to OK.

```
/etc/postgresql/9.3/postgresql.conf
work_mem = 64MB
```

### checkpoint_segments

Jeśli wykonywanych jest dużo operacji na bazie, rozsądne jest zwiększenie parametru checkpoint_segments.

```
checkpoint_segments = 9
checkpoint_timeout = 15min
```

### Autovacuum naptime

Autoodkurzanie w zasadzie powinno się wykonywać jak najczęściej, jednakże bez indywidualnych ustawień dla poszczególnych tabel lepiej zadziała wydłużenie tego czasu.

```
autovacuum_naptime = 5min
```

### max_stack_depth

Maksymalny rozmiar stosu funkcji wykonywanych przez serwer, podany w bajtach. Wartość domyślna to 2048, jednak może okazać się to zbyt mało (np. dla złożonych funkcji rekurencyjnych).

Dokumentacja sugeruje aby wynosiła ona: wynik polecenia "ulimit -s" - 1024 (np. dla ulimit -s = 8192 *max_stack_depth* powinno wynosić 7168)

### pgtune

Dodatkowo polecamy narzędzie PgTune dostępne online http://pgtune.leopard.in.ua/ lub offline https://github.com/gregs1104/pgtune

### pg_xlog

Aby jeszcze trochę przyspieszyć możemy katalog pg_xlog zamontować na partycji bez journal'ingu. Przygotowujemy partycję LVM dla xlog:

```
#lvcreate -L5G -nxlog vg0
```

Tworzymy system plików bez księgowania:

```
#mkfs.ext4 -O ^has_journal /dev/vg0/xlog
```

Zatrzymujemy postgresa

```
#/etc/init.d/postgresql stop
```

Montujemy świeżo utworzoną partycję tymczasowo np:

```
#mount /dev/vg0/xlog /mnt/new
```

Kopiujemy zawartość katalogu pg_xlog do tymczasowej lokalizacji /mnt/new, pamiętając o zachowaniu uprawnień:

```
#rsync -az /var/lib/postgresql/9.3/main/pg_xlog/ /mnt/new/
```

Odmontowujemy partycję xlog z tymczasowej lokalizacji

```
#umount /mnt/new
```

Dodajemy wpis tak aby partycja montowała się razem ze startem systemy. Wpis do fstab:

```
/dev/mapper/vg0-xlog    /var/lib/postgresql/9.3/main/pg_xlog    ext4    defaults,noatime,nodiratime,data=writeback,barrier
```

Montujemy i sprawdzamy poprawność zapisów w fstab :

```
#mount -a
```

Jeśli podłączyła się partycja prawidłowo możemy wystartować postgresa:

```
#/etc/init.d/postgresql start
```

### Pliki bazy na osobnej partycji

. Przygotowujemy partycję LVM dla xlog:

```
#lvcreate -L100G -nbase vg0
```

Tworzymy system plików:

```
#mkfs.ext4 /dev/vg0/base
```

Zatrzymujemy postgresa

```
#/etc/init.d/postgresql stop
```

Montujemy świeżo utworzoną partycję tymczasowo np:

```
#mount /dev/vg0/base /mnt/newbase
```

Kopiujemy zawartość katalogu /var/lib/postgresql do tymczasowej lokalizacji /mnt/newbase, pamiętając o zachowaniu uprawnień:

```
#rsync -az /var/lib/postgresql/ /mnt/newbase/
```

Odmontowujemy partycję newbase z tymczasowej lokalizacji

```
#umount /mnt/newbase
```

Dodajemy wpis tak aby partycja montowała się razem ze startem systemy. Wpis do fstab:

```
/dev/mapper/vg0-base    /var/lib/postgresql    ext4    defaults    0 0
```

Montujemy i sprawdzamy poprawność zapisów w fstab :

```
#mount -a
```

Jeśli podłączyła się partycja prawidłowo możemy wystartować postgresa:

```
#/etc/init.d/postgresql start
```

### Logowanie długich zapytań

```
vim /etc/postgresql/9.3/postgresql.conf
log_min_duration_statement = 2000;
# oznacza logowanie zapytań dłuższych niż 2 sec
```