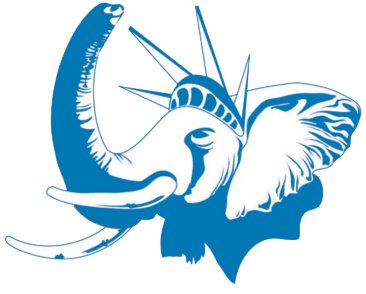
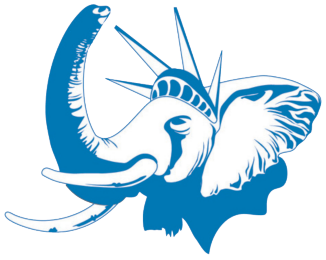
How does PostgreSQL work with disks:
a DBA's checklist in detail



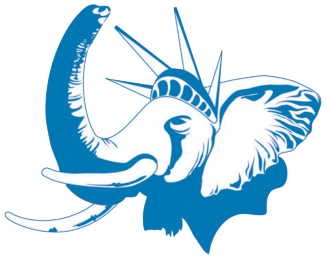
Ilya Kosmodemiansky
ik@postgresql-consulting.com

PostgreSQL-Consulting.com



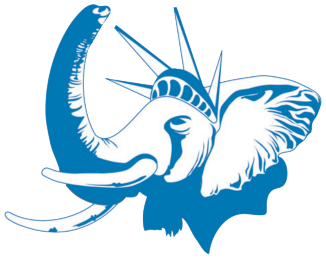
Outline

- Why a database needs disk?
- PostgreSQL specific disk issues
- Bottlenecks
- Monitoring disk subsystem
- Choosing hardware for PostgreSQL
- Configuration tuning



Why a database needs disk?

- To read pages from disk
- To write the Write Ahead Log (WAL)
- To sync WAL with datafiles (CHECKPOINT)

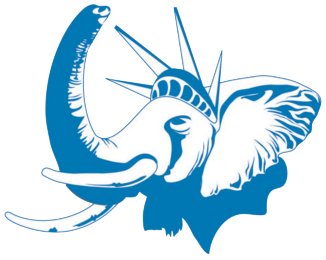


Why a database needs disk?

- To read pages from disk
- To write the Write Ahead Log (WAL)
- To sync WAL with datafiles (CHECKPOINT)

PostgreSQL specifics

- autovacuum
- *pg_clog*
- tmp, disk sorts, hashing

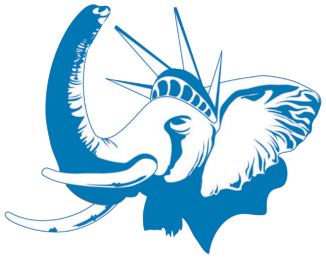


Why a database needs disk?

- To read pages from disk
- To write the Write Ahead Log (WAL)
- **To sync WAL with datafiles (CHECKPOINT)**

PostgreSQL specifics

- autovacuum
- *pg_clog*
- tmp, disk sorts, hashing

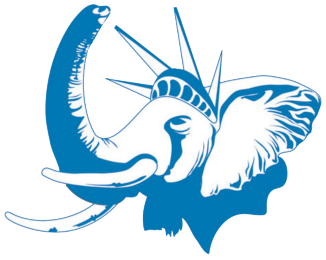


Page lifecycle in PostgreSQL

shared_buffers

operating system cache

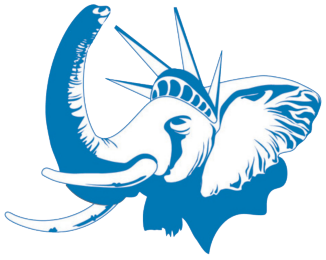
disks



Checkpoint

Why we need checkpoints?

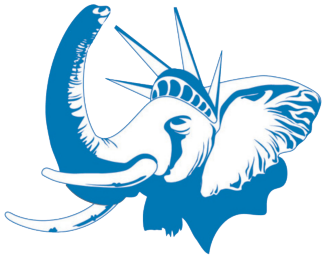
- Database reads "clean"pages into *shared_buffers*; if at least one tuple changed, the page becomes "dirty"
- *COMMIT;* returns, when pages that became dirty in a transaction were synced to WAL
- From time to time the database issues *CHECKPOINT*: dirty pages from *shared_buffers* start beeng synced to disk (fsync)
- Periodical checkpointing makes recovery faster: we need to make undo and redo only until checkpoint
- However, with large *shared_buffers* disc performance during checkpoint can be an issue



Checkpoint

Diagnostics

- Disc utilization spikes on graphical monitoring (iostat -d -x 1, last column %*util*)
- *pg_stat_bgwriter*



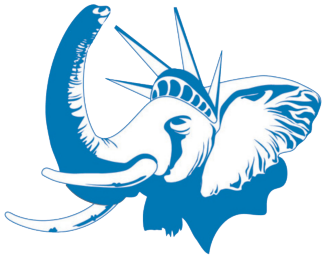
Monitoring

At least

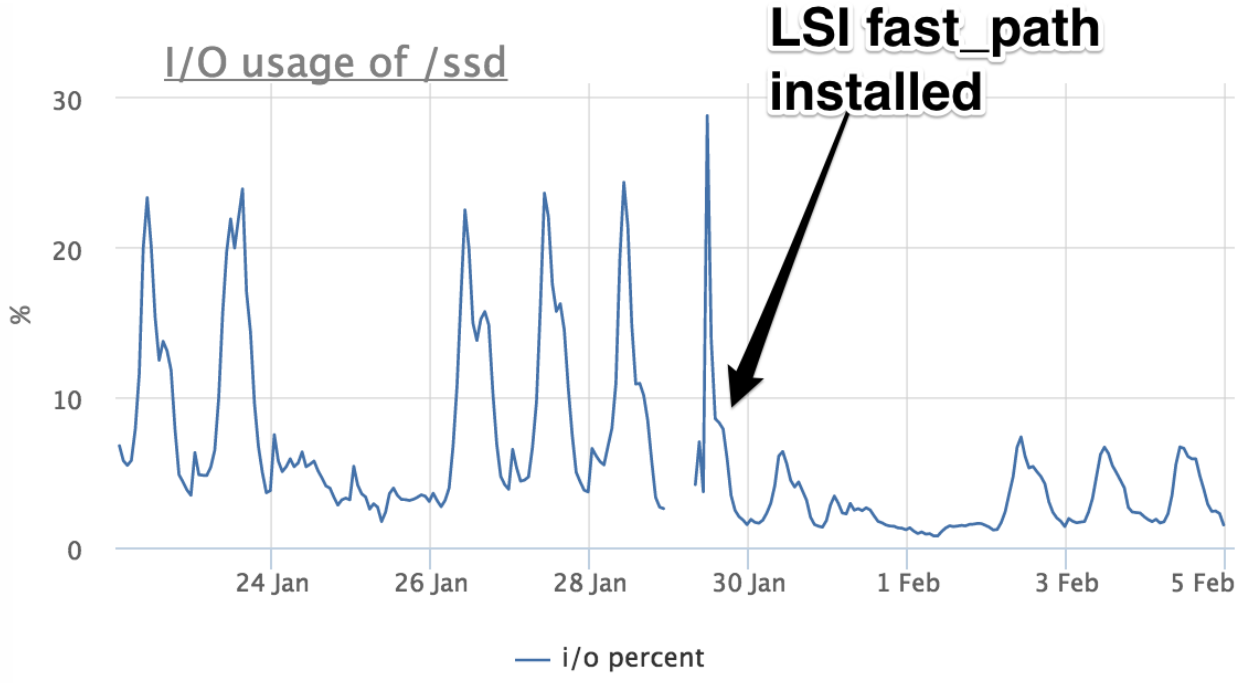
- IOPS - practically useless when it is the only metric
- % utilization
- latency

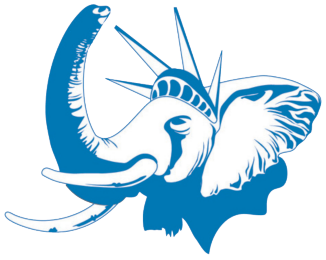
Nice to have

- iowait
- Mbps



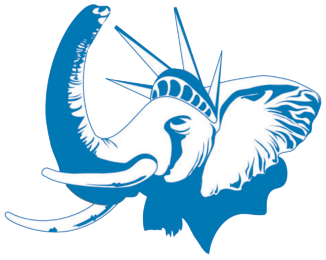
Graph monitoring allows you to see the trend





pg_stat_bgwriter

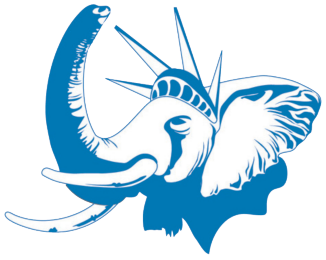
```
pgbench=# select * from pg_stat_bgwriter ;
-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 29
checkpoints_req       | 13
checkpoint_write_time | 206345
checkpoint_sync_time  | 9989
buffers_checkpoint    | 67720
buffers_clean         | 1046
maxwritten_clean      | 0
buffers_backend       | 48142
buffers_backend_fsync | 0
buffers_alloc         | 30137
stats_reset           | 2014-10-24 17:59:15.812002-04

postgres=# select pg_stat_reset_shared('bgwriter');
-[ RECORD 1 ]--------+-
pg_stat_reset_shared |
```



pg_stat_bgwriter

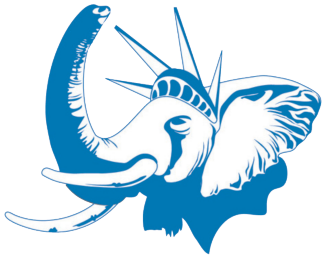
```
pgbench=# select * from pg_stat_bgwriter ;
-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 29
checkpoints_req       | 13
checkpoint_write_time | 206345
checkpoint_sync_time  | 9989
buffers_checkpoint    | 67720
buffers_clean         | 1046
maxwritten_clean      | 0
buffers_backend       | 48142
buffers_backend_fsync | 0
buffers_alloc         | 30137
stats_reset           | 2014-10-24 17:59:15.812002-04

postgres=# select pg_stat_reset_shared('bgwriter');
-[ RECORD 1 ]--------+-
pg_stat_reset_shared |
```

This is a bad (untuned) pg_stat_bgwriter

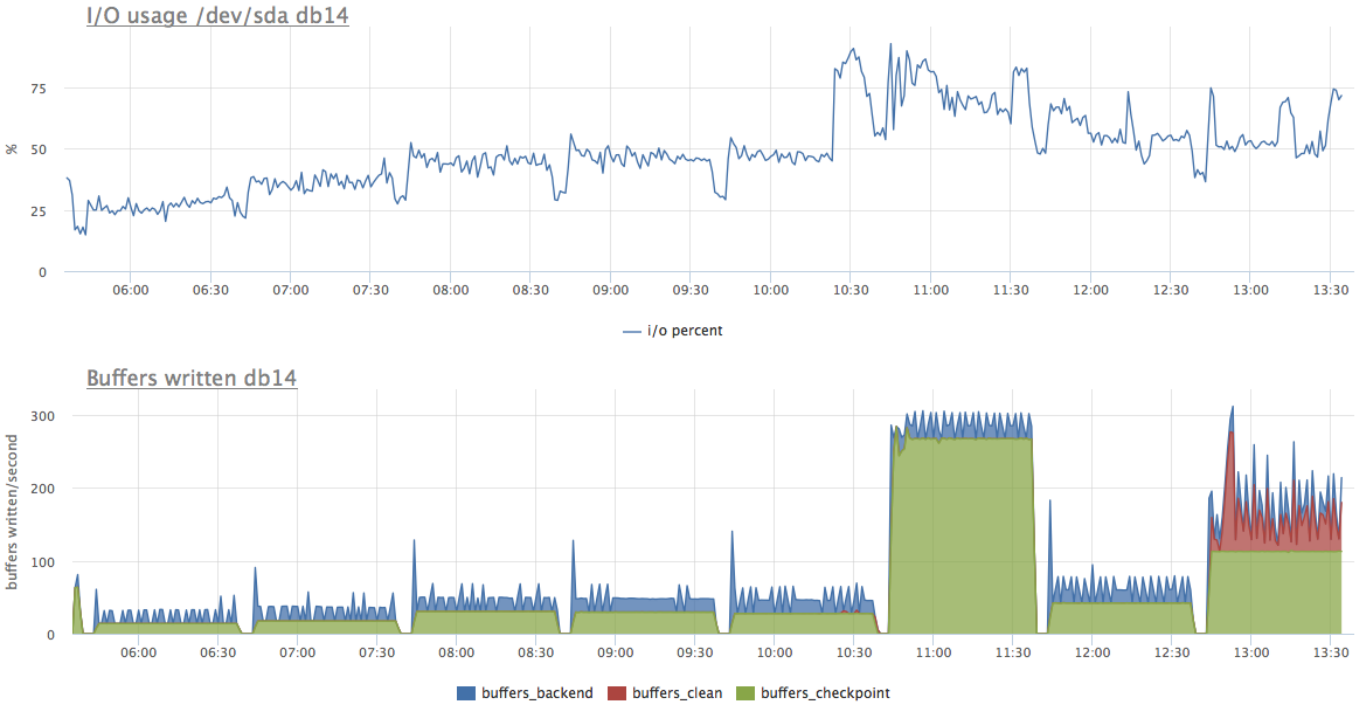


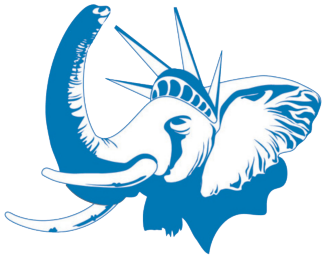
pg_stat_bgwriter - a better one

```
postgres=# select *, now() from pg_stat_bgwriter ;
-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 0
checkpoints_req       | 38
checkpoint_write_time | 20288693
checkpoint_sync_time  | 34751
buffers_checkpoint    | 9176173
buffers_clean         | 0
maxwritten_clean      | 0
buffers_backend       | 10521857
buffers_backend_fsync | 0
buffers_alloc         | 9815168
stats_reset           | 2015-03-22 06:00:02.601286+03
now                   | 2015-03-22 16:01:21.3482+03
```



Nice to have both on one page

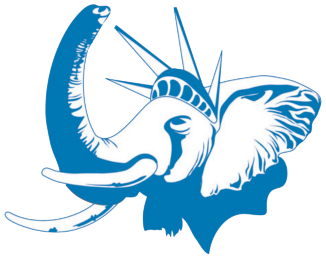




How to make things better?

Hardware: RAID

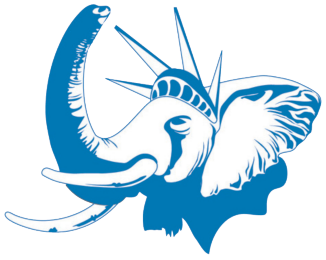
- Cheap RAID controller is worse than a software RAID
- RAID **must** have BBU if we talk about write performance
- Manufacturers LSI or Dell (megaraid or perc) - OK; HP or ARECA have some issues
- Battery should be in a good condition
- cache mode $\rightarrow$ write back
- io mode $\rightarrow$ direct
- Disk Write Cache Mode $\rightarrow$ disabled



How to make things better?

Hardware: disk drives

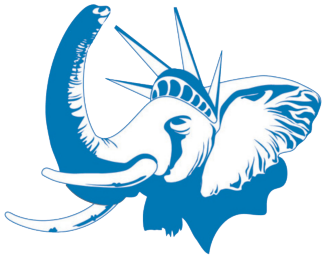
- 2,5"SAS (there are 15K disks, too): 2-3 times faster seek than 3,5"
- No all SSD are good for database: enterprise level Intel p3700 vs desktop-level Samsung
- It is a good idea to use SSDs for your OLTP PostgreSQL installation, but using only SSDs can have drawbacks
- RAID 1+0
- If you cannot afford good discs and RAID-controller *synchronous_commit* $\rightarrow$ *off* can be an option



How to make things better?

Filesystems

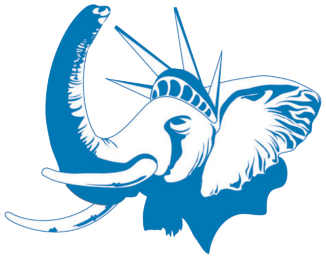
- xfs or ext4: OK
- zfs or any lvm layer are convinient, but it is not the first choise when performance is important
- barrier=0, noatime



How to make things better?

Operating system

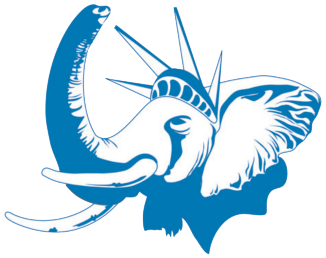
- Defaults in many linux distributives *vm.dirty_ratio = 20* *vm.dirty_background_ratio = 10* - utmost mad
- Much better *vm.dirty_background_bytes = 67108864* *vm.dirty_bytes = 536870912* (512Mb BBU on RAID)
- If no BBU on RAID, values should be devided by 4



How to make things better?

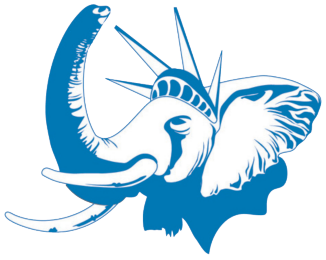
postgresql.conf

- *wal_buffers* (768kB → 16Mb)
- *checkpoint_segments* (3 - checkpoint every 48Mb → 256 - 4Gb)
- *checkpoint_timeout* = 60 (what ever comes first)
- *checkpoint_completion_target* = 0.9 (to spread disk load between checkpoints)



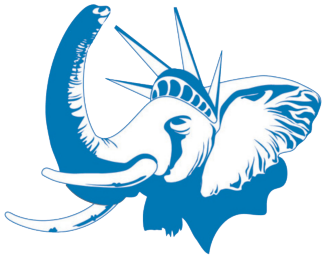
How to check yourself about hardware and OS configu

```
pgdev@pg-dev-deb:~$ tt_pg/bin/pg_test_fsync
5 seconds per test
O_DIRECT supported on this platform for open_datasync and open_sync.

Compare file sync methods using one 8kB write:
(in wal_sync_method preference order, except fdatasync
is Linux's default)
        open_datasync                  11396.056 ops/sec      88 usecs/op
        fdatasync                      11054.894 ops/sec      90 usecs/op
        fsync                          10692.608 ops/sec      94 usecs/op
        fsync_writethrough                   n/a
        open_sync                         67.045 ops/sec   14915 usecs/op

Compare file sync methods using two 8kB writes:
(in wal_sync_method preference order, except fdatasync
is Linux's default)
        open_datasync                   5824.917 ops/sec     172 usecs/op
        fdatasync                      10563.427 ops/sec      95 usecs/op
        fsync                          10234.010 ops/sec      98 usecs/op
        fsync_writethrough                   n/a
        open_sync                         31.837 ops/sec   31410 usecs/op

Compare open_sync with different write sizes:
(This is designed to compare the cost of writing 16kB
in different write open_sync sizes.)
         1 * 16kB open_sync write         62.499 ops/sec   16000 usecs/op
         2 *  8kB open_sync writes        31.248 ops/sec   32002 usecs/op
         4 *  4kB open_sync writes        15.628 ops/sec   63989 usecs/op
```



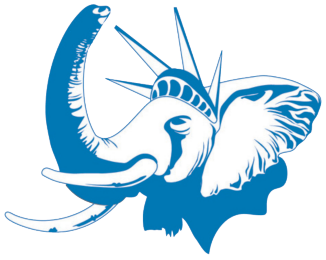
Small hint: let bgwriter do its work

```
postgres=# select name, setting, context, max_val, min_val  from pg_settings
where name ~ 'bgwr';

      name            | setting | context | max_val | min_val
------------------------+---------+---------+---------+---------
 bgwriter_delay         | 200     | sighup  | 10000   | 10
 bgwriter_lru_maxpages  | 100     | sighup  | 1000    | 0
 bgwriter_lru_multiplier | 2       | sighup  | 10      | 0
(3 rows)
```



Do not forget autovacuum

- Bloat makes your database larger
- The more pages involved in a checkpoint, the more slower it is
- autovacuum workers consume IO

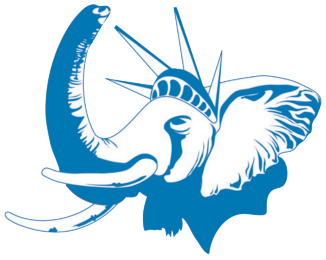


autovacuum: aggressive enough

```
postgres=# select name, setting, context  from pg_settings
where category ~ 'Autovacuum';
```

name	setting	context
autovacuum	on	sighup
autovacuum_analyze_scale_factor	0.05	sighup
autovacuum_analyze_threshold	50	sighup
autovacuum_freeze_max_age	200000000	postmaster
autovacuum_max_workers	10	postmaster
autovacuum_multixact_freeze_max_age	400000000	postmaster
autovacuum_naptime	60	sighup
autovacuum_vacuum_cost_delay	20	sighup
autovacuum_vacuum_cost_limit	-1	sighup
autovacuum_vacuum_scale_factor	0.01	sighup
autovacuum_vacuum_threshold	50	sighup

(11 rows)



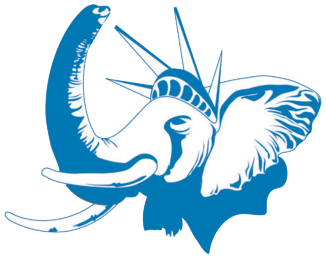
Sometimes a good idea

in crontab:

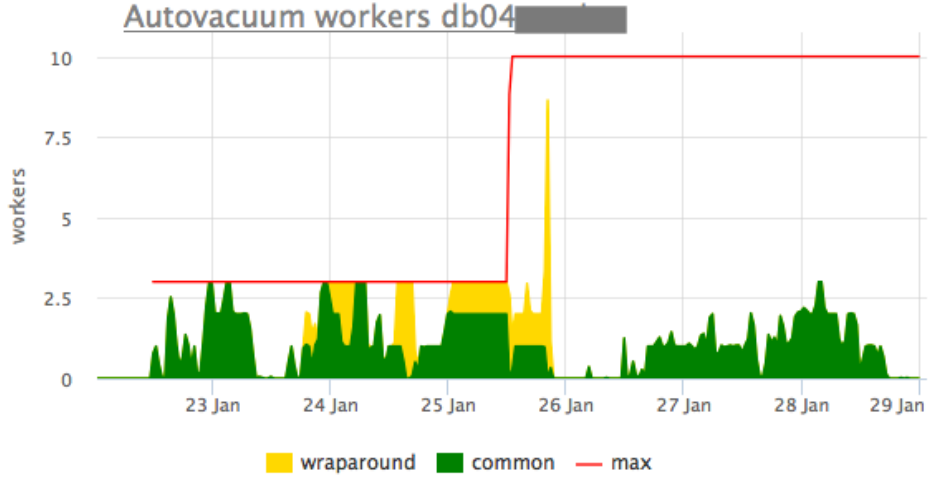
```
* * * * *  /usr/bin/pgrep -f 'postgres: autovacuum' | xargs --no-run-if-empty -I $ renice -n 20 -p $ >/dev/null 2>/dev/null
* * * * *  /usr/bin/pgrep -f 'postgres: autovacuum' | xargs --no-run-if-empty -I $ ionice -c 3 -t -p $
```

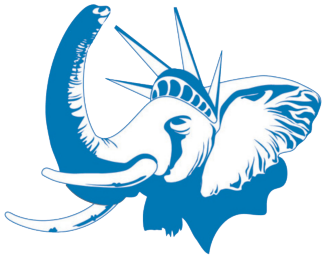
in postgresql.conf:

autovacuum_max_workers → 10-20



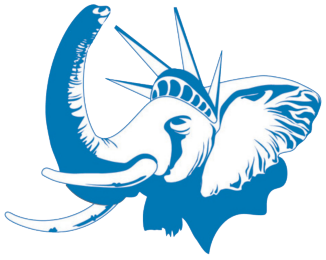
As a result


Autovacuum workers db04
10
7.5
5
2.5
0
workers
23 Jan
24 Jan
25 Jan
26 Jan
27 Jan
28 Jan
29 Jan
wraparound
common
max



Thanks

- To our clients, who provide us with a lot of tricky cases
- To my collegues, who solve them every day
- To the team of http://okmeter.io/ for smart graphics



Thanks

- To our clients, who provide us with a lot of tricky cases
- To my collegues, who solve them every day
- To the team of http://okmeter.io/ for smart graphics

Questions?

ik@postgresql-consulting.com