linuxpolska

File system and virtual memory tuning for a Zabbix database

Alicja Kucharczyk
Senior Solution Architect

Overview

- Why and what for?
- Data
- Methods
- Theoretical background
- Results

hardware

The Hardware

- After an interesting customer’s case (probably NUMA dependent) decided to do my own tests
- it’s NUMA (*Non-uniform memory access*) so I needed at least 4 sockets
- A hosting? Really a few options for 4 sockets & quite expensive
- So decided to buy my own Server

The Hardware

- HP Proliant DL580 G7
- CPU: 4 x Intel® Xeon® Processor X7542 (18M Cache, 2.67 GHz, 6.40 GT/s Intel® QPI)
- RAM: 128 GB DDR3 (10600R)
- Disks: 4 x 300GB SAS 10 000

linuxpolska

environment

Kernel name: Linux

Kernel release: 3.10.0-862.14.4.el7.x86_64

Kernel version: #1 SMP Wed Sep 26 15:12:11 UTC 2018

Hardware name: x86_64

Processor: x86_64

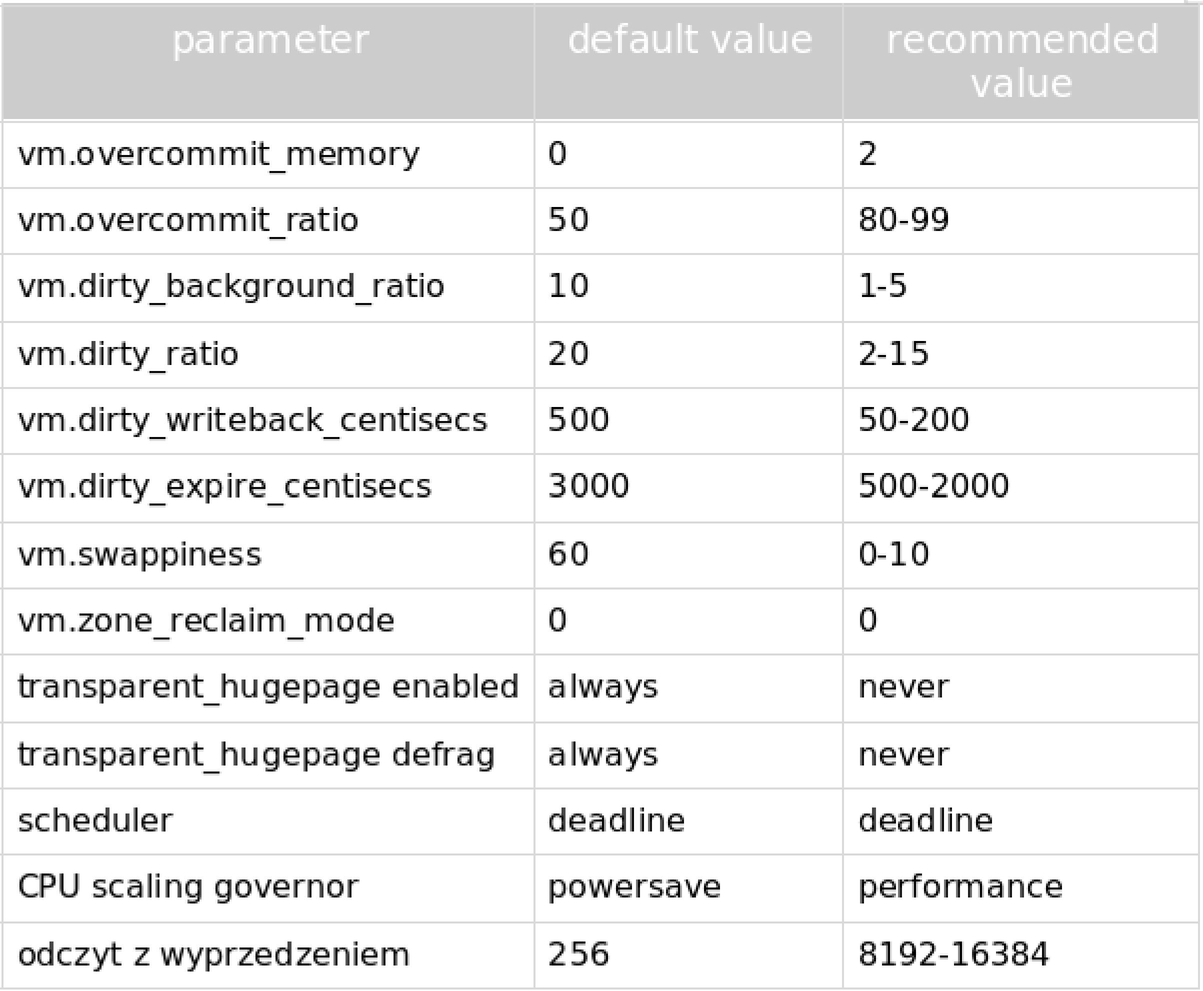
Hardware platform: x86_64

Red Hat release: CentOS Linux release 7.5.1804 (Core)

background

background

- Operating system configuration check is always done during db audits
- Parameters and the „right values” were chosen from a lot of solid sources
- But never investigated in a real production environment



parameter	default value	recommended value
vm.overcommit_memory	0	2
vm.overcommit_ratio	50	80-99
vm.dirty_background_ratio	10	1-5
vm.dirty_ratio	20	2-15
vm.dirty_writeback_centisecs	500	50-200
vm.dirty_expire_centisecs	3000	500-2000
vm.swappiness	60	0-10
vm.zone_reclaim_mode	0	0
transparent_hugepage enabled	always	never
transparent_hugepage defrag	always	never
scheduler	deadline	deadline
CPU scaling governor	powersave	performance
odczyt z wyprzedzeniem	256	8192-16384

data

- But where to get those „real data” from?
- Fortunately one of our customer agreed to use their data for these tests
- Because of this in the title of this presentation you can find Zabbix

ZABBIX

data

Production:

- ~4TB of data
- A big polish public institution
- Data from tens of thousands metrics
- 1 PostgreSQL 10 instance with 1 hot standby

data extraction

Preparations:

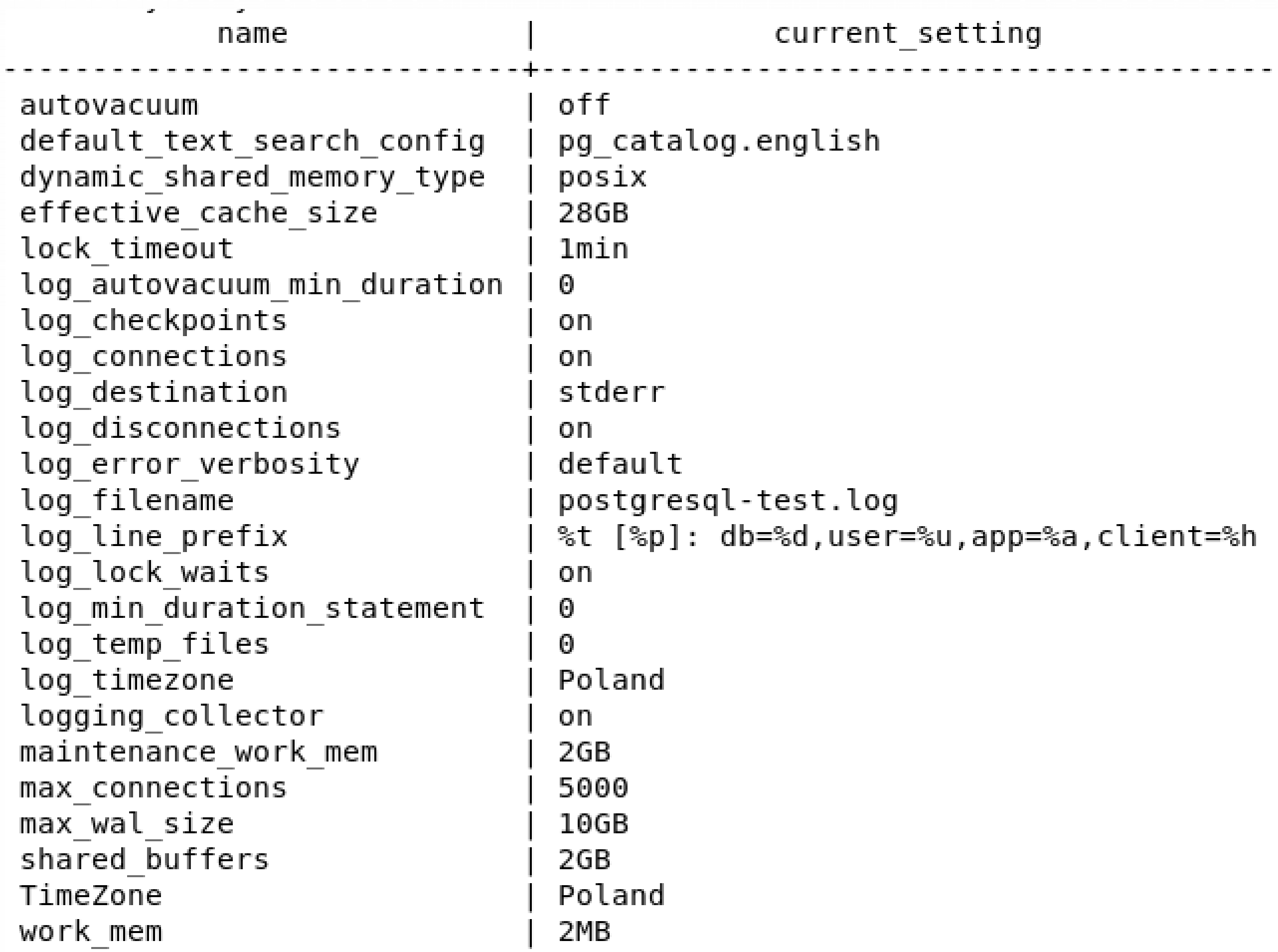
- DB logical snapshot (*pg_dump*)
- Text logs (not WAL’s) gathered for 2 days since snapshot was taken
- log_min_duration_statement = 0

methods

Single test run

- duration: 1hour
- rc.local script that starts the test
- a new parameter value is set
- pgreplay starts
- after 1 hour pgreplay process is killed
- reboot

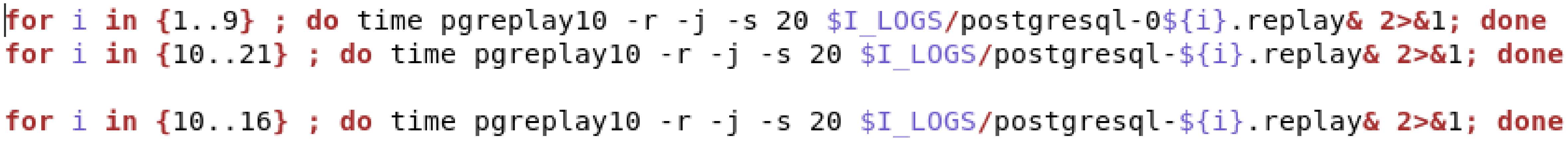
Db configuration



```
            name             |             current_setting
-----------------------------+-----------------------------------------
 autovacuum                  | off
 default_text_search_config  | pg_catalog.english
 dynamic_shared_memory_type  | posix
 effective_cache_size        | 28GB
 lock_timeout                | 1min
 log_autovacuum_min_duration | 0
 log_checkpoints             | on
 log_connections             | on
 log_destination             | stderr
 log_disconnections          | on
 log_error_verbosity         | default
 log_filename                | postgresql-test.log
 log_line_prefix             | %t [%p]: db=%d,user=%u,app=%a,client=%h
 log_lock_waits              | on
 log_min_duration_statement  | 0
 log_temp_files              | 0
 log_timezone                | Poland
 logging_collector           | on
 maintenance_work_mem        | 2GB
 max_connections             | 5000
 max_wal_size                | 10GB
 shared_buffers              | 2GB
 TimeZone                    | Poland
 work_mem                    | 2MB
```

methods

To increase the load all the logs were replayed at once, some logs were replayed twice:



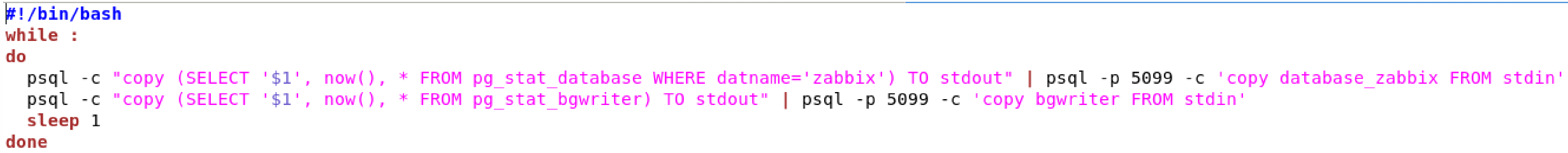
```
for i in {1..9} ; do time pgreplay10 -r -j -s 20 $I_LOGS/postgresql-0${i}.replay& 2>&1; done
for i in {10..21} ; do time pgreplay10 -r -j -s 20 $I_LOGS/postgresql-${i}.replay& 2>&1; done

for i in {10..16} ; do time pgreplay10 -r -j -s 20 $I_LOGS/postgresql-${i}.replay& 2>&1; done
```

methods

Metrics:

- PgBadger
- Data from 2 views written every second to another db



```bash
#!/bin/bash
while :
do
  psql -c "copy (SELECT '$1', now(), * FROM pg_stat_database WHERE datname='zabbix') TO stdout" | psql -p 5099 -c 'copy database_zabbix FROM stdin'
  psql -c "copy (SELECT '$1', now(), * FROM pg_stat_bgwriter) TO stdout" | psql -p 5099 -c 'copy bgwriter FROM stdin'
  sleep 1
done
```

overcommit

Overcommit

There is a lot of programs that request huge amounts of memory "just-in-case" and don't use much of it

The Linux kernel supports the following overcommit handling modes (*overcommit_memory*):

0 - Heuristic overcommit handling (default)

1 - Always overcommit

2 - "never overcommit" policy that attempts to prevent any overcommit of memory

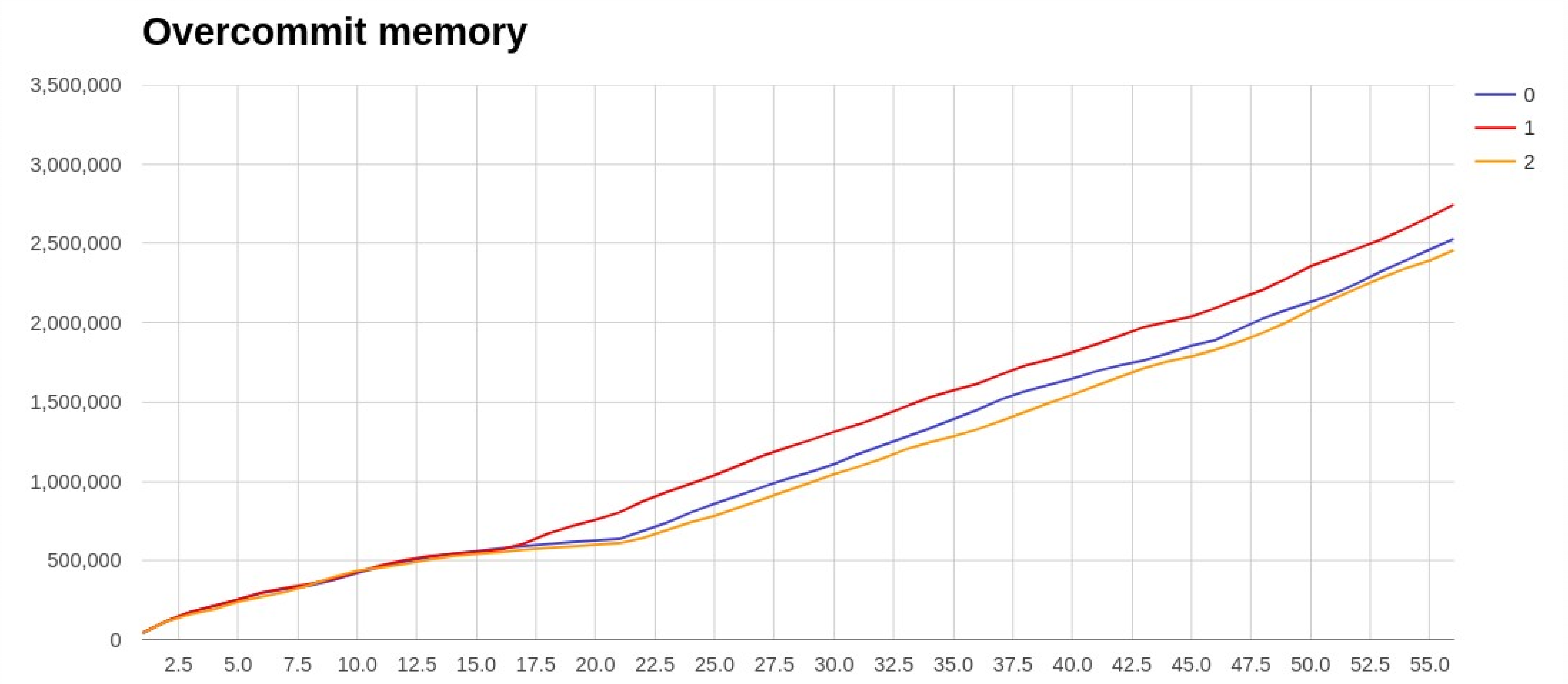
Overcommit

scary movie X

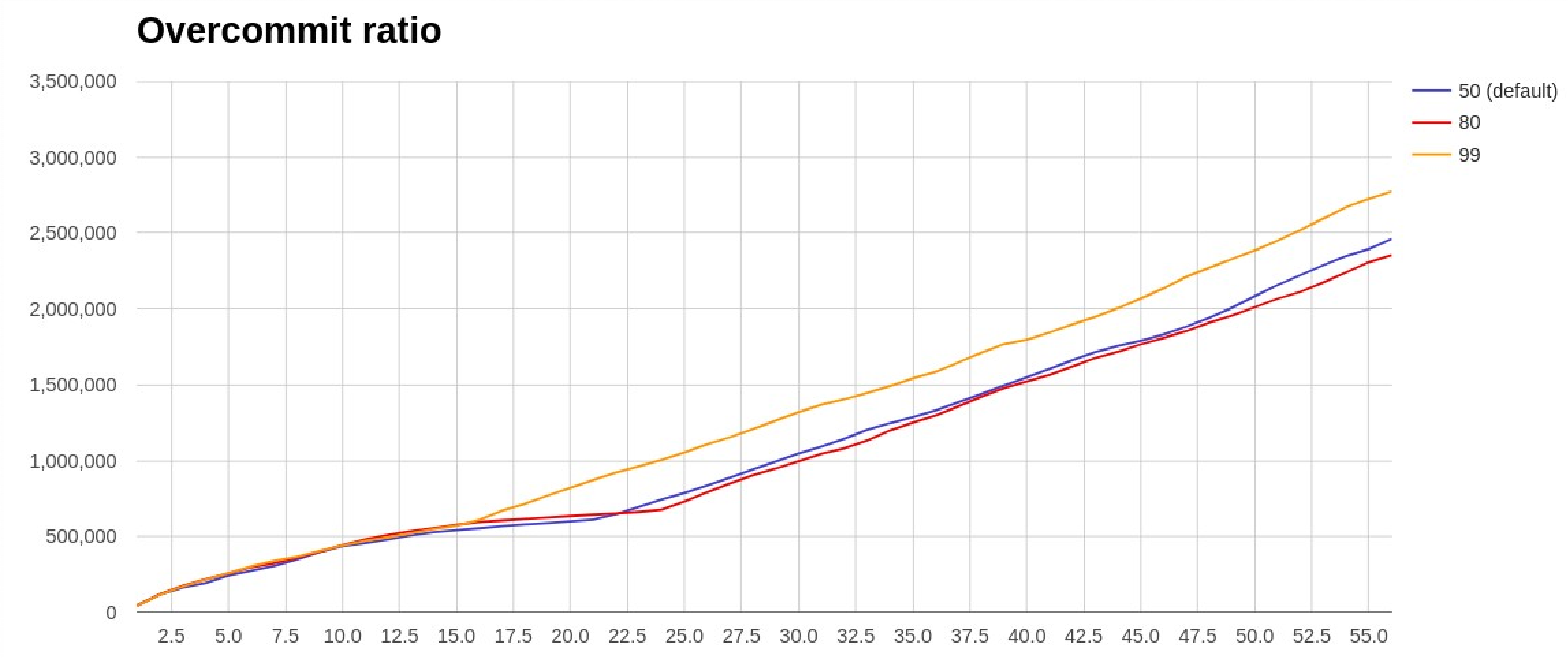
Overcommit

- overcommit_memory - flag that enables memory overcommitment
- overcommit_ratio - when *overcommit_memory* is set to 2 - the total address space commit for the system is not permitted to exceed swap + a configurable amount (default is 50%) of physical RAM

Overcommit memory



Overcommit ratio



writeout of dirty data to disk

writeout of dirty data to disk

Buffered writes - operating system read and write caches are used

Dirty page doesn’t go directly to the disk - it gets flushed to the OS write cache which then writes it to disk

writeout of dirty data to disk

Writeback tuning parameters:

- dirty_background_ratio & dirty_ratio (space)
- dirty_expire_centisecs, dirty_writeback_centisecs (time)

writeout of dirty data to disk

dirty_background_ratio - defines the percentage of memory that can become dirty before a background flushing of the pages to disk starts. Until this percentage is reached no pages are flushed to disk. However when the flushing starts, then **it's done in the background without disrupting any of the running processes in the foreground**. (or *dirty_background_bytes*)

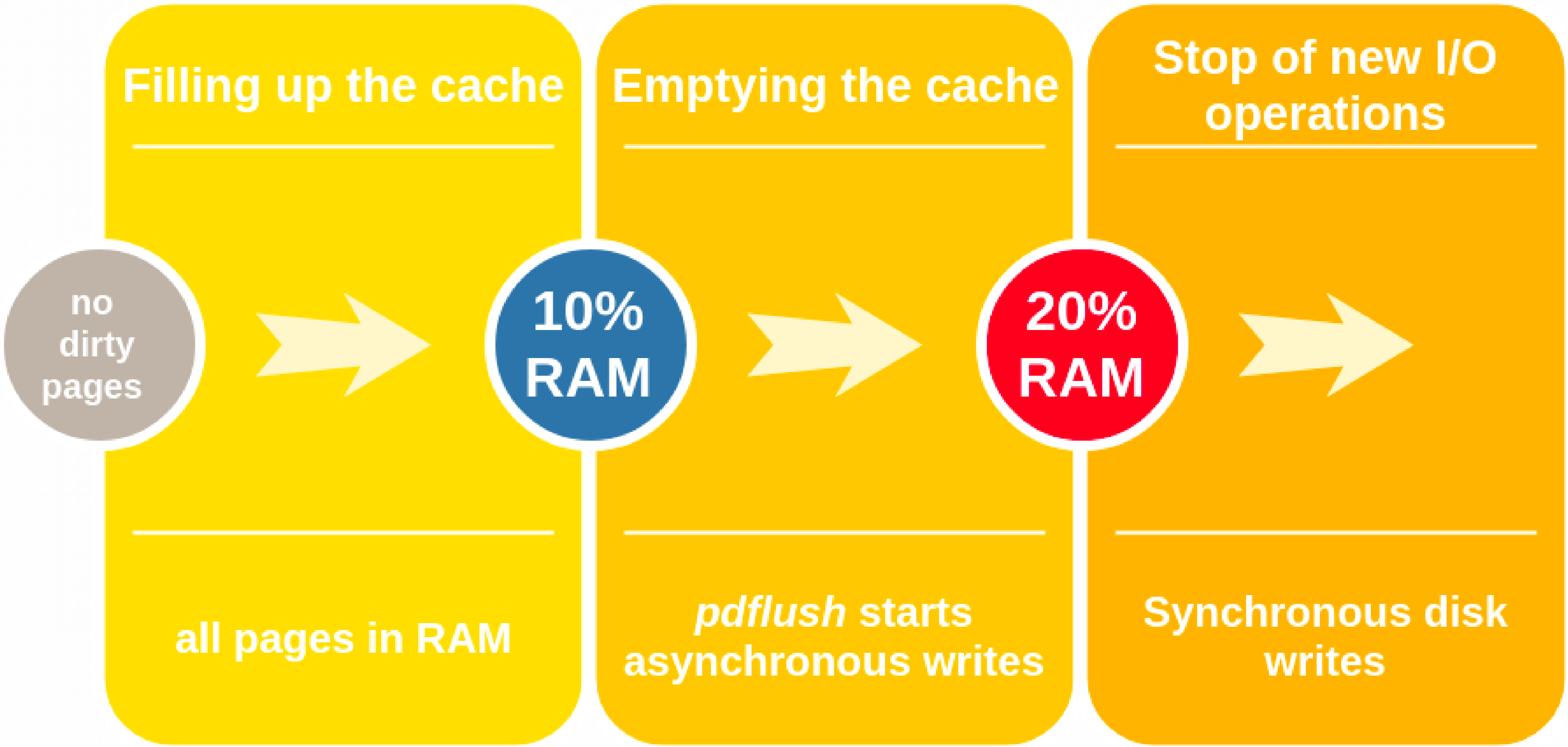
default: 10%

Overcommit

dirty_ratio - defines the percentage of memory which can be occupied by dirty pages before a forced flush starts. If the percentage of dirty pages reaches this number, then **all processes become synchronous**, they are not allowed to continue until the io operation they have requested is actually performed and the data is on disk (or *dirty_bytes*)

default: 20%

writeout of dirty data to disk



no dirty pages

Filling up the cache

all pages in RAM

10% RAM

Emptying the cache

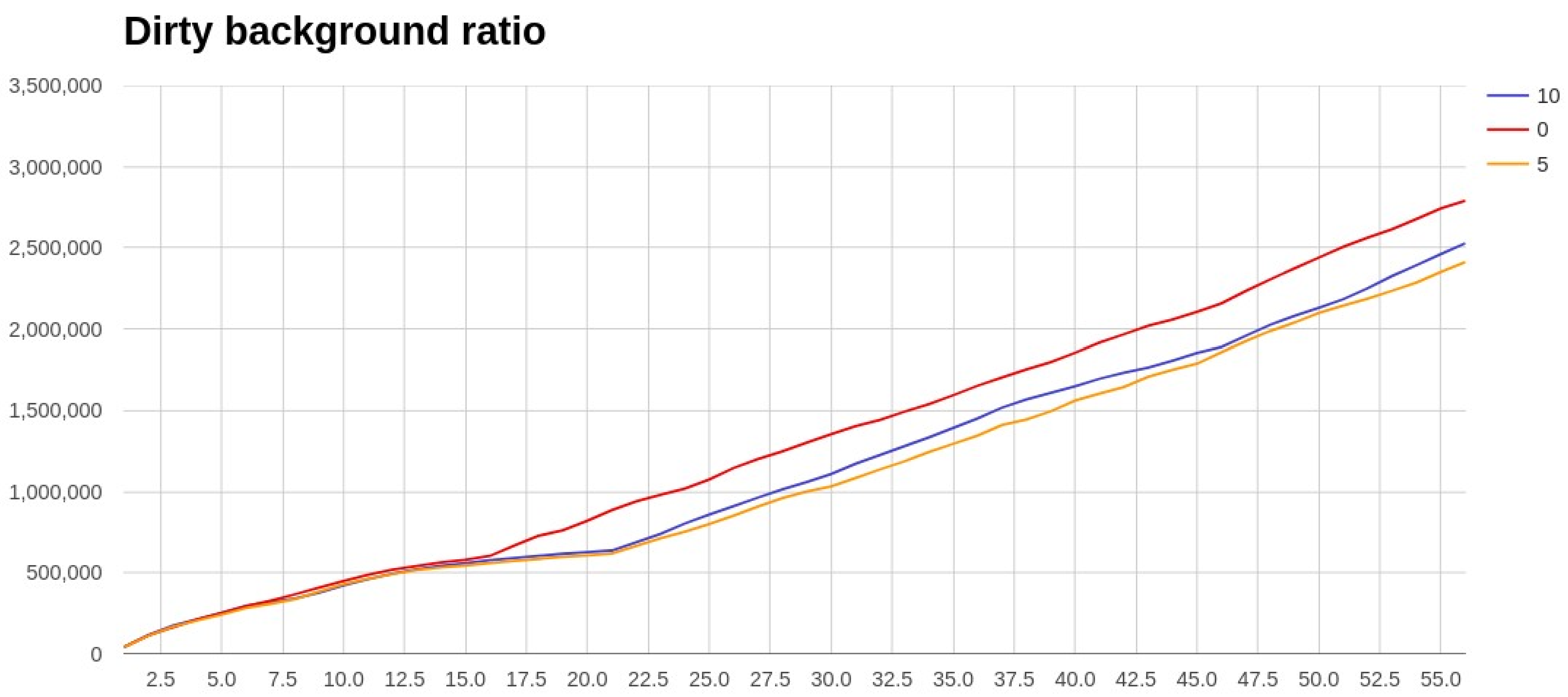
pdflush starts asynchronous writes

20% RAM

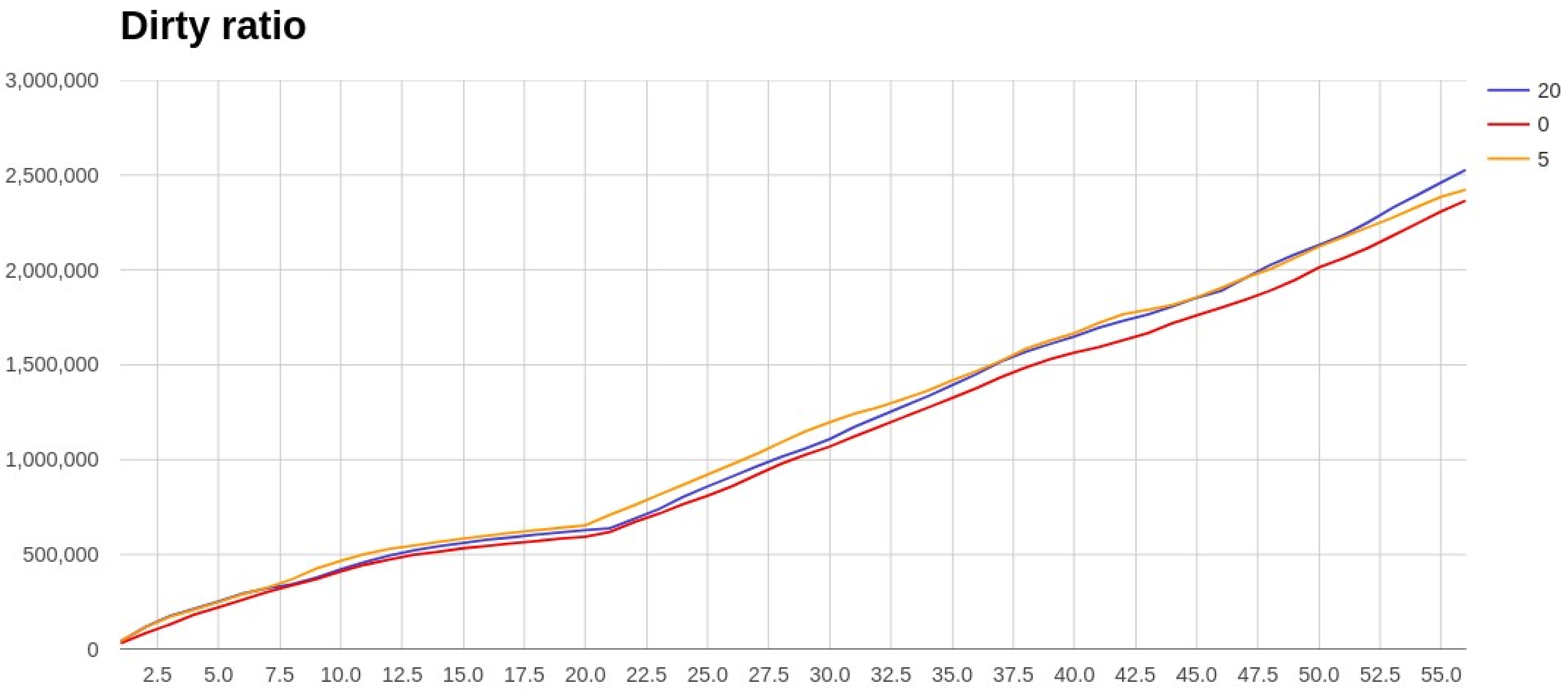
Stop of new I/O operations

Synchronous disk writes

dirty background ratio



dirty ratio


Dirty ratio
3,000,000
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5
55.0
20
0
5

HugePages

HugePages

x86 CPUs usually address memory in 4kB pages, but they are capable of using larger 2 MB or 1 GB pages known as huge pages.

Two kinds of huge pages:

- pre-allocated at startup
- allocated dynamically during runtime

Transparent HugePages

- enabled by default with Red Hat Enterprise Linux 6, Red Hat Enterprise Linux 7, SUSE 11, Oracle Linux 6, and Oracle Linux 7

Transparent HugePages

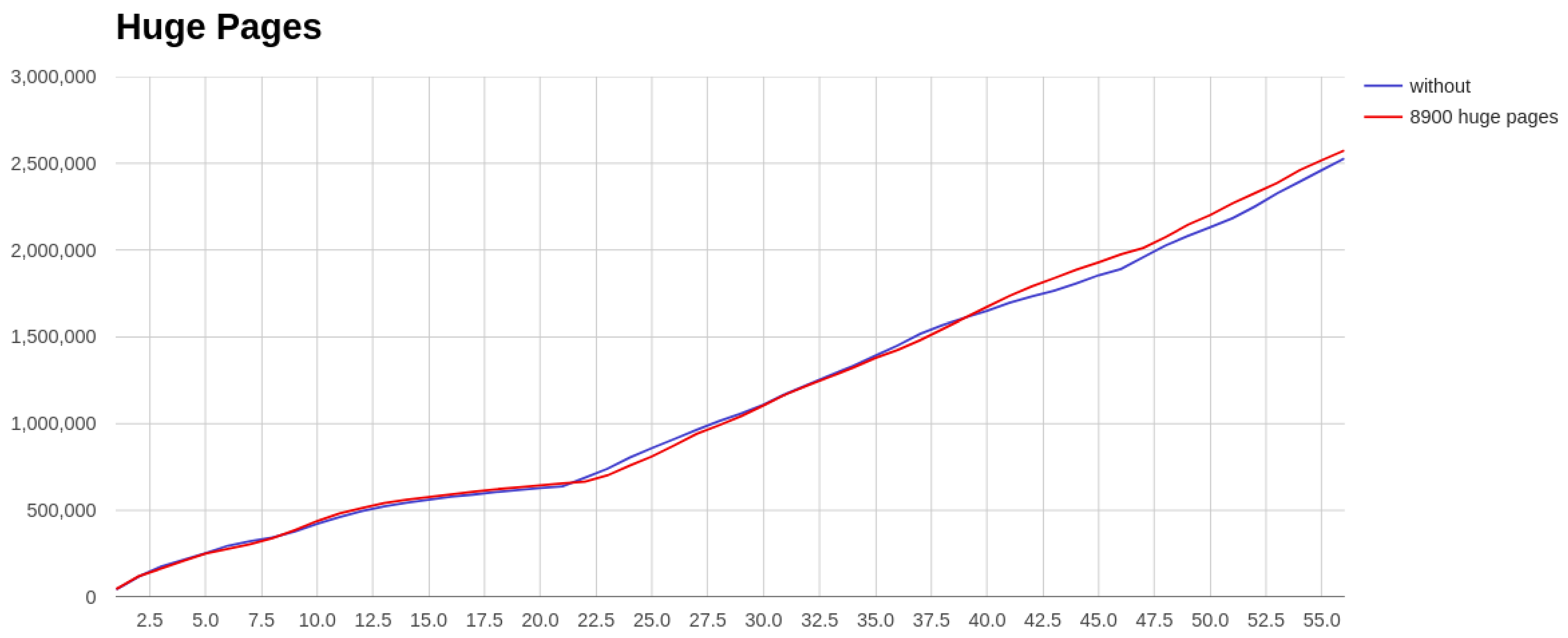
„Oracle recommends that you disable Transparent HugePages before you start installation.”

Release 12.2 Oracle Documentation

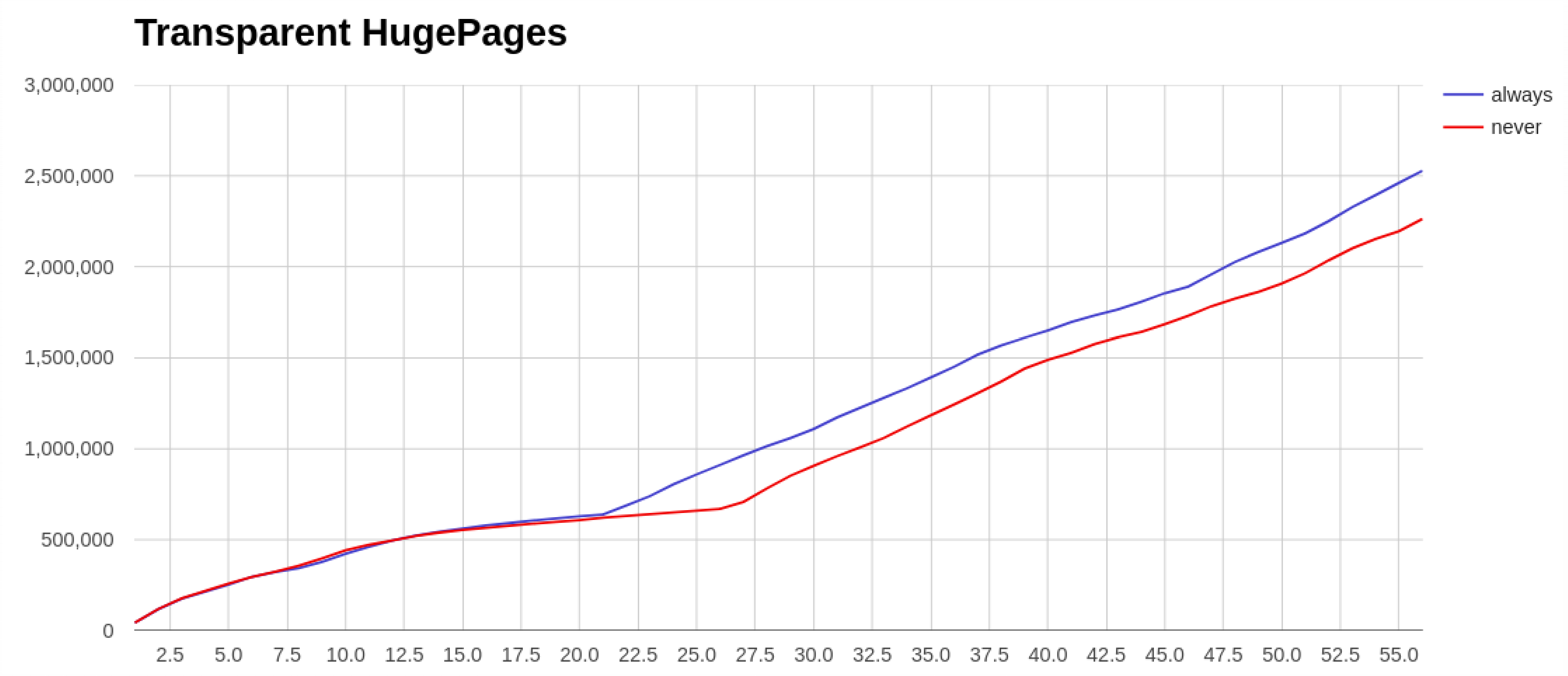
„Disable Transparent Huge Pages (THP)”

MongoDB Documentation

HugePages


Huge Pages
3,000,000
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5
55.0
without
8900 huge pages

Transparent HugePages



read-ahead

linuxpolska

read-ahead

„The first parameter you should tune on any Linux install

is the device read-ahead.”

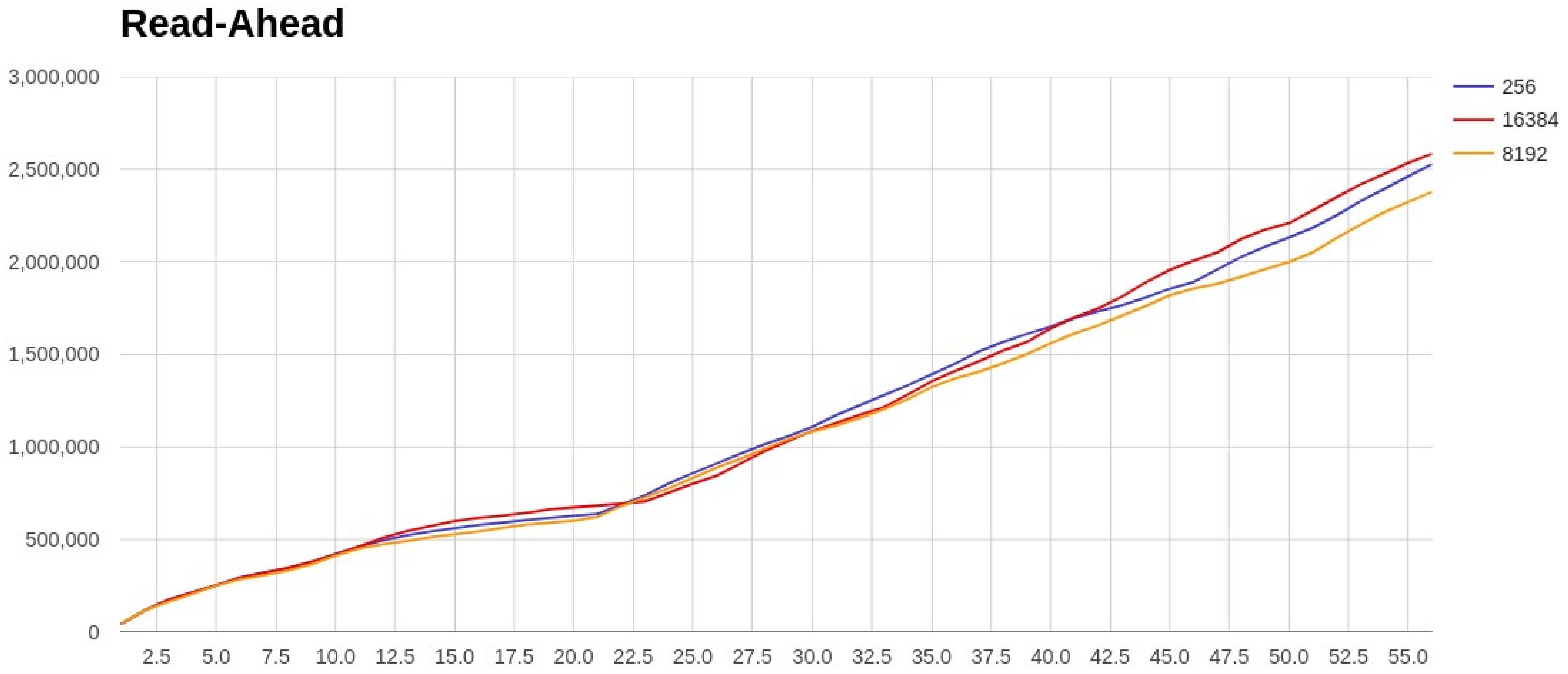
Ibrar Ahmed, Greg Smith

PostgreSQL 9.6 High Performance

read-ahead

Readahead is a system call of the Linux kernel that loads a file's contents into the page cache. This prefetches the file so that when it is subsequently accessed, its contents are read from the main memory (RAM) rather than from a hard disk drive (HDD), resulting in much lower file access latencies.

read-ahead


Read-Ahead
3,000,000
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5
55.0
256
16384
8192

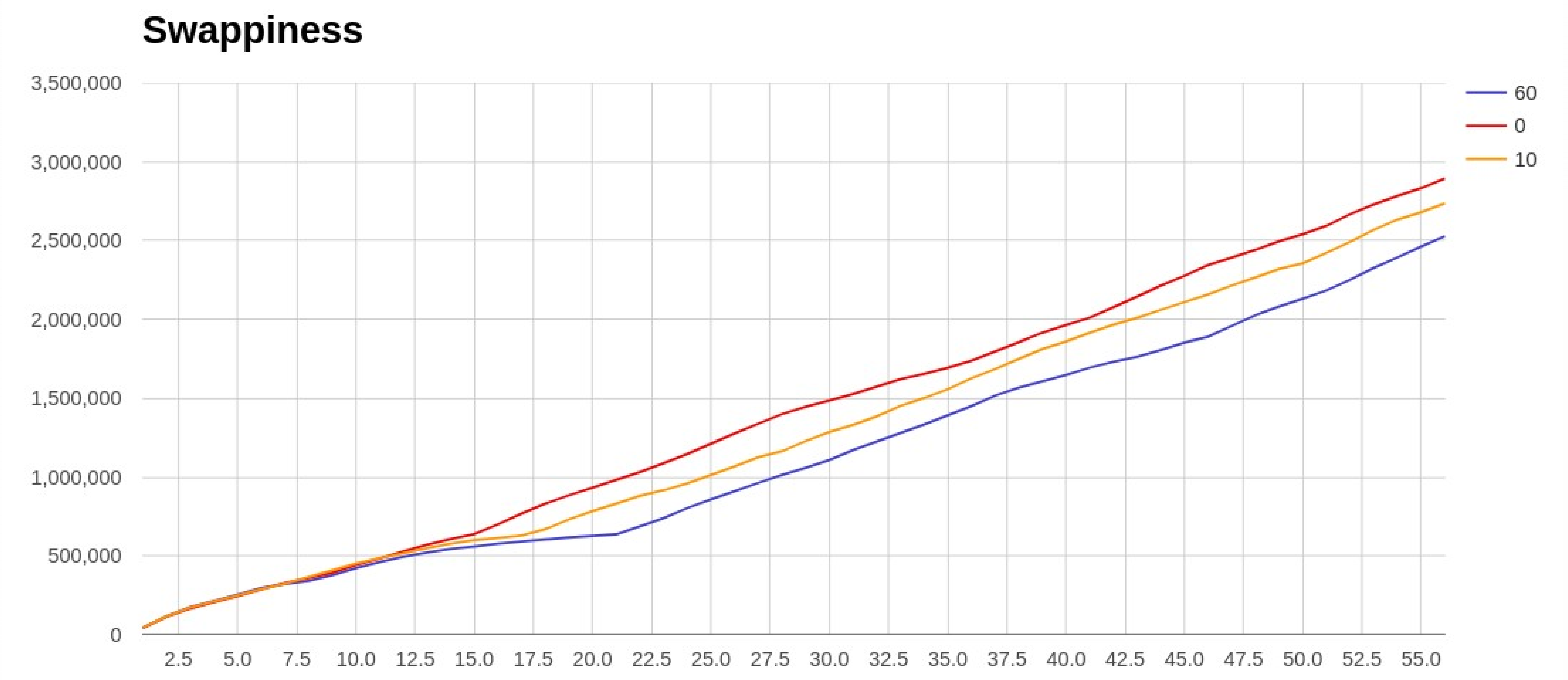
swappiness

swappiness

- controls how much the kernel favors swap over RAM
- higher values will increase aggressiveness
- lower values decrease the amount of swap

default: 60

swappiness



Swappiness
3,500,000
3,000,000
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5
55.0
60
0
10

mount options

noatime

- Do not update access times on this filesystem

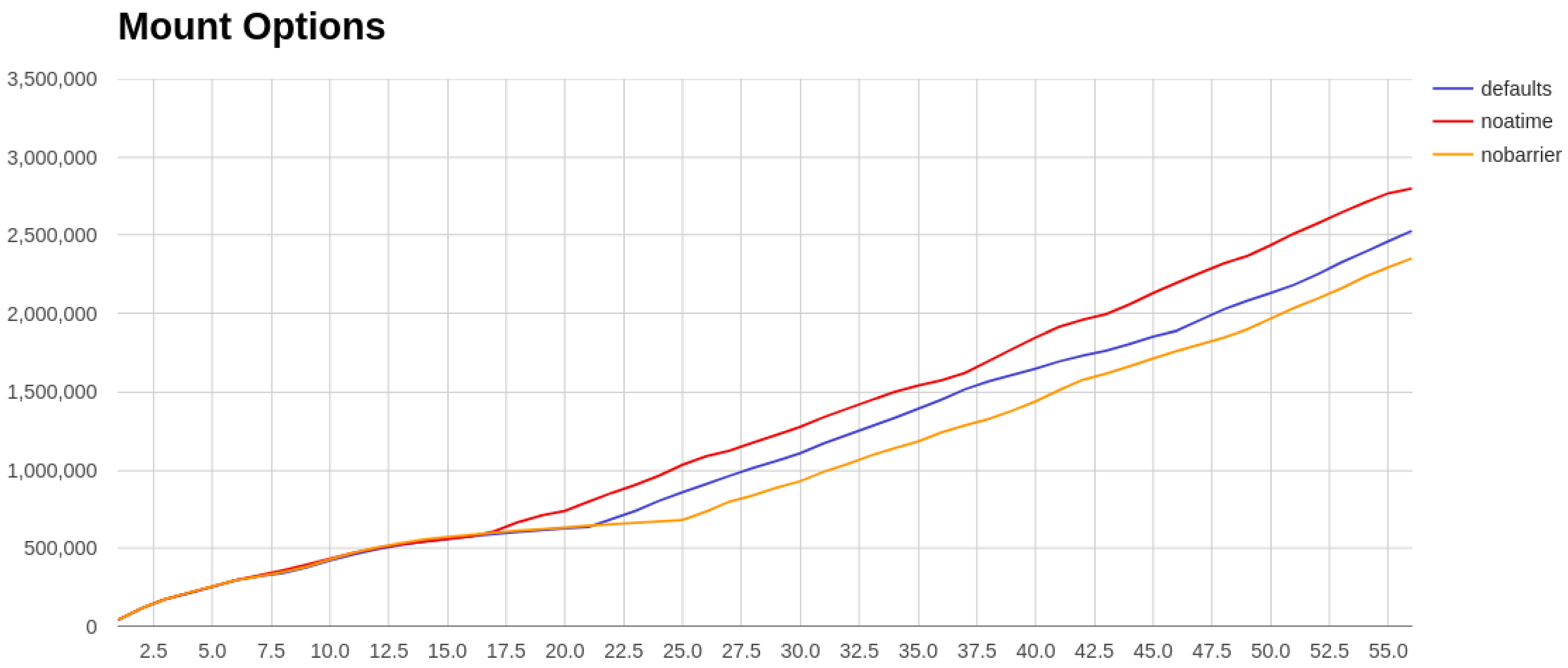
```
/dev/mapper/centos-azot on /azot type xfs (rw,noatime,seclabel,attr2,inode64,noquota)
```

[default value: relatime; recommended: noatime]

noatime

- I/O barriers ensure that requests actually get written to non-volatile medium in order
- filesystem integrity protection when power failure or some other events stop the drive from operating and possibly make the drive lose data in its cache
- nobarrier option disables this feature

noatime


Mount Options
3,500,000
3,000,000
2,500,000
2,000,000
1,500,000
1,000,000
500,000
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5
45.0
47.5
50.0
52.5
55.0
defaults
noatime
nobarrier

I/O schedulers

I/O schedulers

„People seem drawn to this area, hoping that it will have a real impact on the performance of their system, based on the descriptions. The reality is that these are being covered last because this is the least-effective tunable mentioned in this section.”

Ibrar Ahmed, Greg Smith

PostgreSQL 9.6 High Performance

I/O schedulers

- decide in which order the block I/O operations will be submitted to storage volumes
- reorders the incoming randomly ordered requests so the associated data would be accessed with minimal arm/head movement
- noop [deadline] cfq

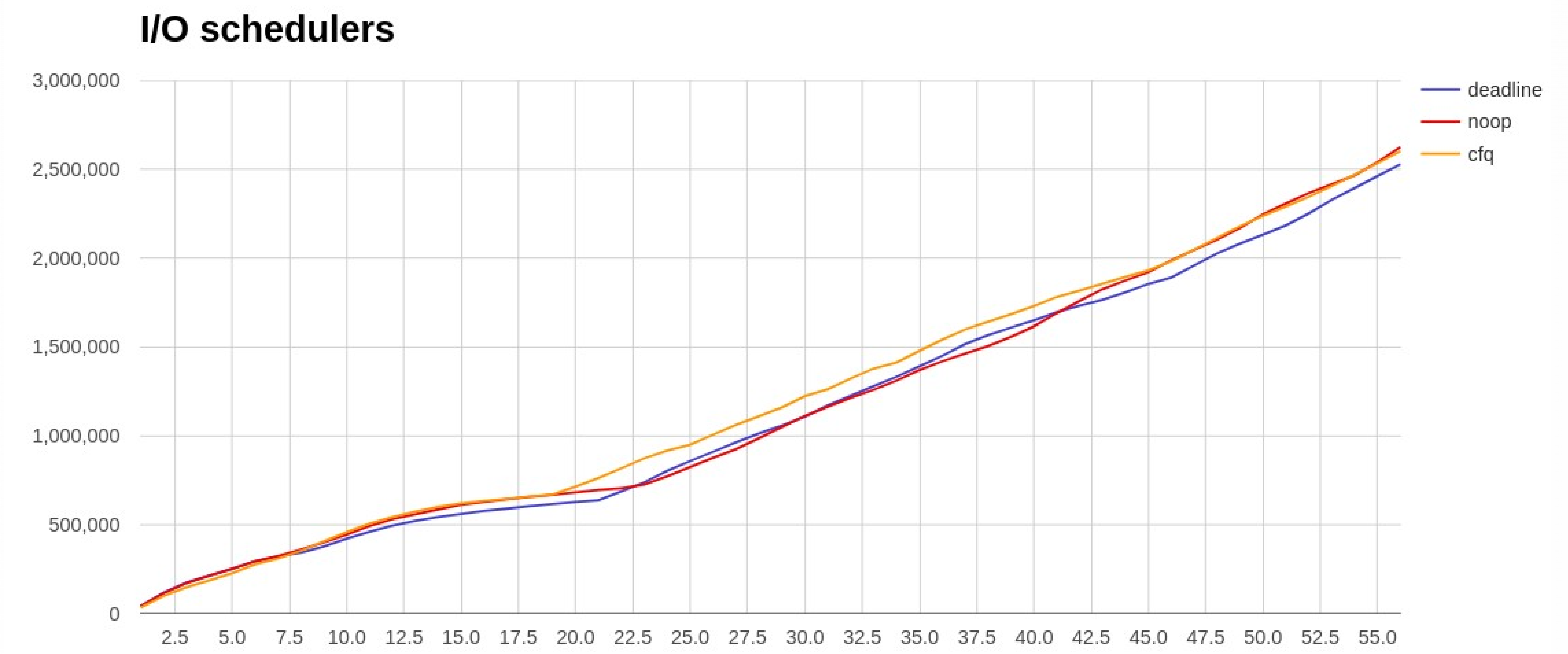
I/O schedulers

„Anyone who tells you that either CFQ or deadline is always the right choice doesn't know what they're talking about”

Ibrar Ahmed, Greg Smith

PostgreSQL 9.6 High Performance

I/O schedulers



separated volumes

separated volumes

„It is advantageous if the log is located on a different disk from the main database files”

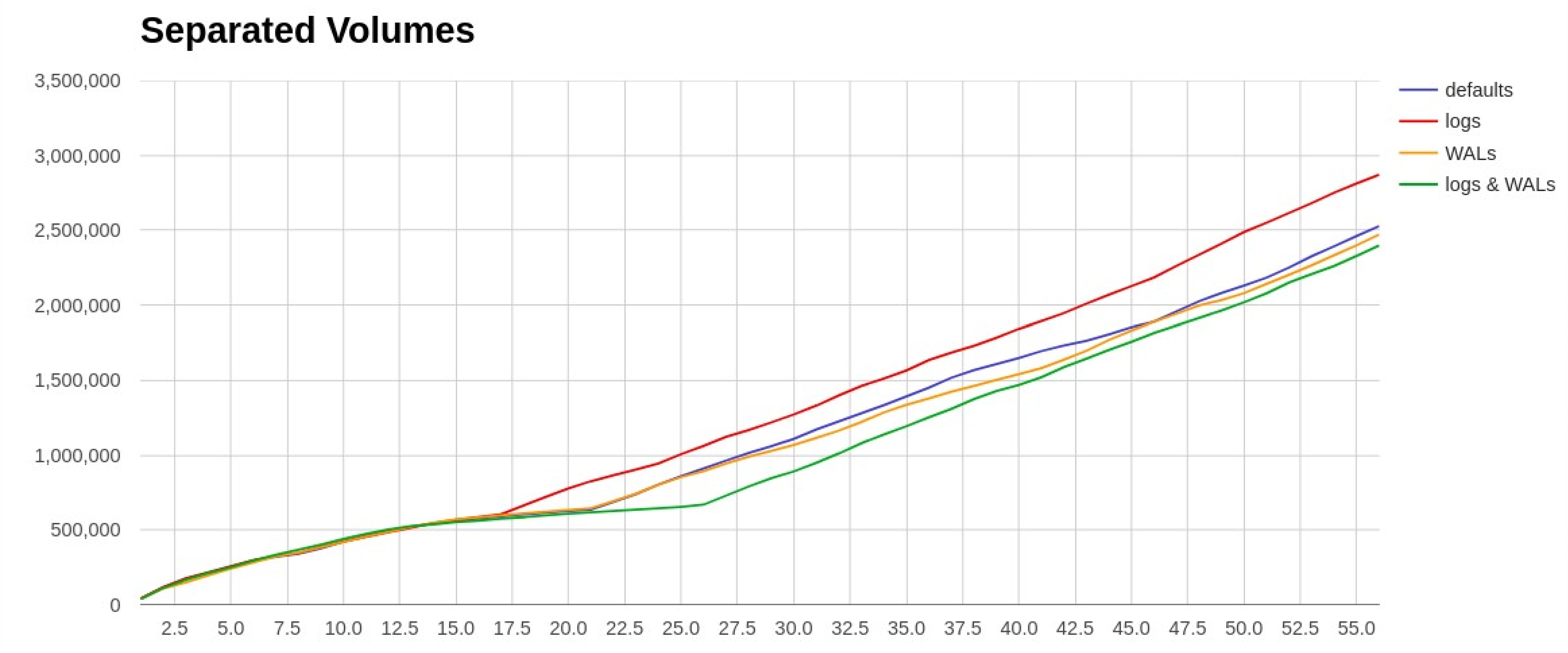
PostgreSQL Documentation

separated volumes

What to separate?

- WALs
- indexes
- temporary files
- temporary statistics data (stats_temp_directory)
- error logs
- highly read or written tables
- [...]

separated volumes



References

- https://www.kernel.org/doc/Documentation/sysctl/vm.txt
- https://www.kernel.org/doc/html/latest/vm/overcommit-accounting.html?highlight=overcommit
- https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/performance_tuning_guide/s-memory-tunables
- https://hep.kbfi.ee/index.php/IT/KernelTuning
- https://en.wikipedia.org/wiki/Readahead
- https://docs.oracle.com/en/database/oracle/oracle-database/12.2/cwlin/disabling-transparent-hugepages.html#GUID-02E9147D-D565-4AF8-B12A-8E6E9F74BEEA
- https://docs.mongodb.com/manual/tutorial/transparent-huge-pages/
- https://en.wikipedia.org/wiki/I/O_scheduling
- https://patchwork.kernel.org/patch/134161/
- https://www.postgresql.org/docs/current/static/index.html

linuxpolska

Thank You!

please leave your feedback on:
https://2018.pgconf.eu/f

Alicja Kucharczyk
Senior Solution Architect
alicja.kucharczyk@linuxpolska.pl
+48 888 700 065