Vindicating ZFS with PostgreSQL: Unleashing the Power of Scalability

POSETTE 2024

Federico Campoli

June 2024

Few words about the speaker

- Born in 1972
- Passionate about IT since 1985
- In love with PostgreSQL since 2006
- PostgreSQL and FreeBSD tattoo on the right shoulder
- Freelance DBA

Getting in touch

KAMEDATA
DEVOPS AND DATA ENGINEERING

- **Linkedin:** https://www.linkedin.com/in/federicocampoli/
- **Github:** https://github.com/the4thdoctor
- **Soundcloud:** https://soundcloud.com/the4thdoctor
- **Blog:** https://pgdba.org

Disclaimer

- This is talk is based on real life experience
- The tests are reenactments
- There are A LOT of topics not covered by this talk
- Always test before going in production

Table of contents

Warning!

The story you are about to hear is true.
Only the names have been changed to protect the innocent.

A long time ago,
in a startup far far away...

There was a PostgreSQL running on someone else's computer.
AKA THE CLOUD!

A long time ago...

Misused and abused by...
THE DEVELOPERS!

A long time ago...

The poor PostgreSQL ended up completely...
BLOATED!

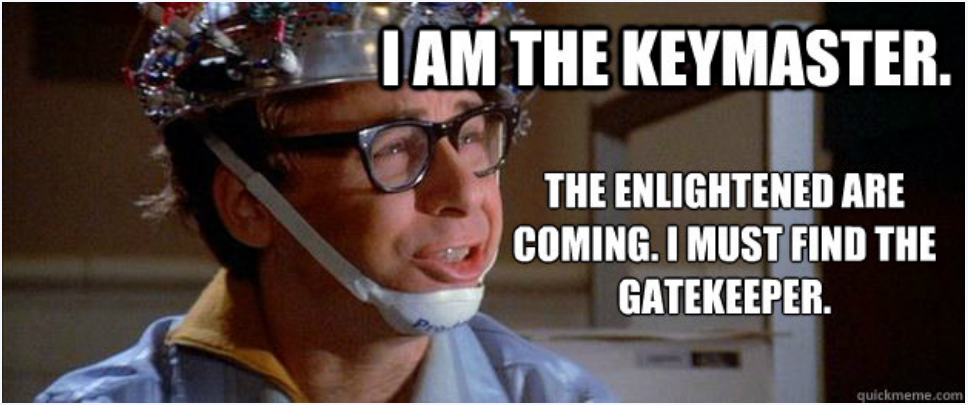
A long time ago...

So they called...
THE DBA!

A long time ago...

The DBA approached the problem delicately as always.

PostgreSQL and Linux are not friends...



Source http://www.quickmeme.com/meme/3s27y1

PostgreSQL

- Multiprocessing
- Block size 8192 bytes
- One big shared memory segment
- Consumes a lot of memory

PostgreSQL and Linux are not friends...

Source https://forum.blu-ray.com/

Linux

- NUMA, memory pages scattered in the RAM
- Max filesystem block size 4096 bytes on X86
- Dirty flush blocks the IO
- OOM killer!

Source https://imgflip.com/i/2uj2dx

PostgreSQL

- Tune the max connections
- Enable the usage of huge pages
- Do not oversize the shared buffer

Linux

- Disable NUMA
- Configure and use the huge pages
- Configure correctly the vm.dirty_ratio
- Disable the memory overcommit
- On supported architectures set the filesystem block size to 8192 bytes
- Or use a filesystem more flexible, like ZFS

Mount options for ext4

- noatime disables updating of access time for files. Implies nodiratime.
- nodiratime disables updating of access time for directories only
- barrier=0 disables the write barriers for the mount point
- data=writeback Sets the data journaling in writeback mode

Do not disable barriers on wal area or on virtual machine's disks. Use database block checksums.

Enter ZFS

Described as "The last word in filesystems", ZFS is stable, fast, secure, and future-proof. Features of ZFS include:

- pooled storage (integrated volume management – zpool)
- Copy-on-write
- snapshots
- data integrity verification and automatic repair (scrubbing)
- RAID-Z
- a maximum 16 exabyte file size
- and a maximum 256 quadrillion zettabyte storage with no limit on number of filesystems (datasets) or files.

Taken shamelessy from https://wiki.archlinux.org/title/ZFS

ZFS Cons

- License CCDL not compatible with the GPL
- Memory greedy
- Maintenance requires attention
- Not very efficient in writes
- Recent bug causing corruption
  https://www.theregister.com/2023/11/27/openzfs_2_2_0_data_corruption/

ZFS options

The advantage of ZFS is that the options can be changed dynamically.
No need to remount the dataset.
Most of the changes apply to newly written data though.

- atime atime can be turned on or off
- recordsize defaults to 128k can be adjusted accordingly
- compression ZFS supports the transparent compression, supports lz4, gzip, zstd
- logbias sets the strategy for the ZIL if present (latency or throughput)

KAMEDATA
DEVOPS AND DATA ENGINEERING

ZFS module can be optimised as well

The parameters can be set as option for the module.

```
# Save the contents in /etc/modprobe.d/zfs.conf
# change PARAMETER for workload XZY to solve problem PROBLEM_DESCRIPTION
# changed by YOUR_NAME on DATE
options zfs PARAMETER=VALUE
```

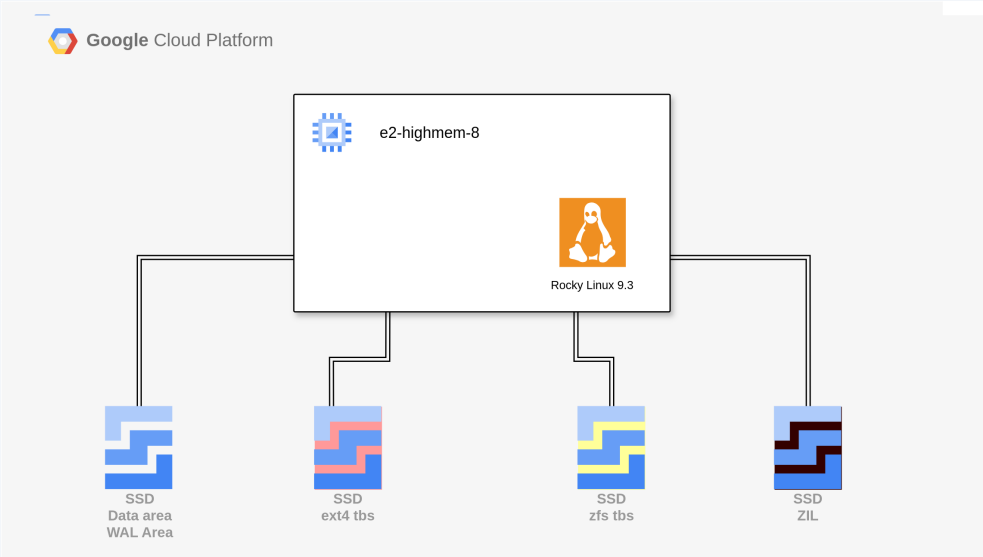
ZFS module can be optimised as well

Most of the parameters can be set dynamically.

```
echo NEWVALUE >> /sys/module/zfs/parameters/PARAMETER
```

Let's do some tests!

The system used for tests



The system used for tests

- "Hardware" Google Compute Engine e2-highmem-8
- 62 GB installed RAM
- 8 processors AMD EPYC 7B12
- separate volumes for wal area and data area on ssd single device
- ext4 tablespace on persistent disk ssd
- zfs tablespace on persistent disk ssd
- zil persistent on persistent disk ssd
- no swap area
- OS Rocky Linux release 9.3 (Blue Onyx)

Kernel configuration

- vm.dirty_background_ratio=5
- vm.dirty_ratio=99
- vm.overcommit_memory=2
- vm.overcommit_ratio=100
- vm.swappiness=0

PostgreSQL setup

- PostgreSQL 16.2
- shared_buffers 1 GB
- work_mem 400 MB
- max_connections 100
- max_wal_size 5GB

pgbench

- transaction type: builtin: TPC-B (sort of)
- scaling factor: 500
  - pgbench_accounts 50,000,000 rows
  - pgbench_branches 500 rows
  - pgbench_tellers 5000 rows
- number of clients: 50
- number of threads: 1
- Duration: 120 seconds
- No foreign keys
- pgbench initialisation for each each test

Example

```
dropping old tables...
creating tables...
generating data (client-side)...
100000 of 50000000 tuples (0%) done (elapsed 0.08 s, remaining 41.26 s)
.
.
.
50000000 of 50000000 tuples (100%) done (elapsed 65.88 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 85.74 s
(drop tables 0.10 s, create tables 0.02 s, client-side generate 66.26 s, vacuum 0.64
     s, primary keys 18.71 s).
```

Space used roughly 7.4 GB

ext4 default settings run

Example

```
starting vacuum...end. pgbench (16.2)
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 500
query mode: simple
number of clients: 50
number of threads: 1
maximum number of tries: 1
duration: 120 s
number of transactions actually processed: 586545
number of failed transactions: 0 (0.000%)
latency average = 10.176 ms
initial connection time = 712.894 ms
tps = 4913.281291 (without initial connection time)
```

Example

Remounted ext4 with the options noatime,nodiratime

```
dropping old tables...
creating tables...
generating data (client-side)...
100000 of 50000000 tuples (0%) done (elapsed 0.08 s, remaining 37.76 s)
.
.
.
vacuuming...
creating primary keys...
done in 102.09 s (drop tables 0.00 s, create tables 0.01 s, client-side generate
    80.91 s, vacuum 1.17 s, primary keys 20.00 s).
```

Space used roughly 7.5 GB

ext4 run with noatime,nodiratime

Example

```
starting vacuum...end. pgbench (16.2)
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 500
query mode: simple
number of clients: 50
number of threads: 1
maximum number of tries: 1
duration: 120 s
number of transactions actually processed: 641218
number of failed transactions: 0 (0.000%)
latency average = 9.309 ms
initial connection time = 674.030 ms
tps = 5371.271157 (without initial connection time)
```

Example

Remounted ext4 with the options noatime,nodiratime,data=writeback

```
dropping old tables...
creating tables...
generating data (client-side)...
100000 of 50000000 tuples (0%) done (elapsed 0.08 s, remaining 40.65 s)
.
.
.
50000000 of 50000000 tuples (100%) done (elapsed 72.02 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 94.21 s (drop tables 0.07 s, create tables 0.02 s, client-side generate
    72.89 s, vacuum 0.80 s, primary keys 20.43 s).
```

Space used roughly 7.5 GB

ext4 run with noatime,nodiratime,data=writeback

Example

```
starting vacuum...end. pgbench (16.2)
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 500
query mode: simple
number of clients: 50
number of threads: 1
maximum number of tries: 1
duration: 120 s
number of transactions actually processed: 648422
number of failed transactions: 0 (0.000%)
latency average = 9.204 ms
initial connection time = 691.884 ms
tps = 5432.483119 (without initial connection time)
```

zfs initialisation with no tuning

Example

```
dropping old tables...
creating tables...
generating data (client-side)...
100000 of 50000000 tuples (0%) done (elapsed 0.08 s, remaining 39.27 s)
.
.
.
50000000 of 50000000 tuples (100%) done (elapsed 61.51 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 90.32 s (drop tables 0.00 s, create tables 0.01 s, client-side generate
    61.77 s, vacuum 1.44 s, primary keys 27.10 s).
```

Space used roughly 7.4 GB

zfs run with no tuning

Example

```
starting vacuum...end. pgbench (16.2)
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 500
query mode: simple
number of clients: 50
number of threads: 1
maximum number of tries: 1
duration: 120 s
number of transactions actually processed: 214904
number of failed transactions: 0 (0.000%)
latency average = 28.146 ms
initial connection time = 691.692 ms
tps = 1776.425893 (without initial connection time)
```

Pretty bad isn't it?

KAMEDATA
DEVOPS AND DATA ENGINEERING

- ext4 no tuning: 4913 TPS
- ext4 noatime: 5371 TPS
- **ext4 noatime,writeback: 5432 TPS**
- **zfs no tuning: 1776 TPS (WHAT THE...!?!?)**

Example

Similarly to ext4 we can turn atime off for the dataset.

```
dropping old tables...
creating tables...
generating data (client-side)...
100000 of 50000000 tuples (0%) done (elapsed 0.08 s, remaining 40.91 s)
.
.
.
50000000 of 50000000 tuples (100%) done (elapsed 61.35 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 88.74 s (drop tables 0.11 s, create tables 0.01 s, client-side generate
    61.56 s, vacuum 1.39 s, primary keys 25.67 s).
```

Space used roughly 7.4 GB

Example

```
starting vacuum...end. pgbench (16.2)
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 500
query mode: simple
number of clients: 50
number of threads: 1
maximum number of tries: 1
duration: 120 s
number of transactions actually processed: 221130
number of failed transactions: 0 (0.000%)
latency average = 27.249 ms
initial connection time = 684.563 ms
tps = 1834.921537 (without initial connection time)
```

A slight improvement

- ext4 no tuning: 4913 TPS
- ext4 noatime: 5371 TPS
- **ext4 noatime,writeback: 5432 TPS**
- zfs no tuning: 1776 TPS
- **zfs atime off: 1835 TPS (Still WHAT THE...!?!?)**

Example

We can enable the compression using the lz4 algorithm.

```
dropping old tables...
creating tables...
generating data (client-side)...
100000 of 50000000 tuples (0%) done (elapsed 0.08 s, remaining 41.57 s)
.
.
50000000 of 50000000 tuples (100%) done (elapsed 51.81 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 74.02 s (drop tables 0.13 s, create tables 0.01 s, client-side generate
    52.02 s, vacuum 1.44 s, primary keys 20.43 s).
```

Space used roughly 986 MB

zfs with atime off, lz4 compression

Example

```
starting vacuum...end. pgbench (16.2)
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 500
query mode: simple
number of clients: 50
number of threads: 1
maximum number of tries: 1
duration: 120 s
number of transactions actually processed: 509697
number of failed transactions: 0 (0.000%)
latency average = 11.714 ms
initial connection time = 692.930 ms
tps = 4268.286775 (without initial connection time)
```

A good jump

- ext4 no tuning: 4913 TPS
- ext4 noatime: 5371 TPS
- **ext4 noatime,writeback: 5432 TPS**
- zfs no tuning: 1776 TPS
- zfs atime off: 1835 TPS
- **zfs atime off compression lz4: 4268 TPS**
  Space used 986 MB

Example

Reduce the record size to 16k

```
dropping old tables...
creating tables...
generating data (client-side)...
100000 of 50000000 tuples (0%) done (elapsed 0.07 s, remaining 36.26 s)
.
.
50000000 of 50000000 tuples (100%) done (elapsed 56.71 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 81.98 s (drop tables 0.10 s, create tables 0.01 s, client-side generate
    56.93 s, vacuum 0.78 s, primary keys 24.16 s).
```

Space used roughly 2.2 GB

Example

```
starting vacuum...end. pgbench (16.2)
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 500
query mode: simple
number of clients: 50
number of threads: 1
maximum number of tries: 1
duration: 120 s
number of transactions actually processed: 750717
number of failed transactions: 0 (0.000%)
latency average = 7.952 ms
initial connection time = 683.204 ms
tps = 6288.119711 (without initial connection time)
```

Now we are talking

- ext4 no tuning: 4913 TPS
- ext4 noatime: 5371 TPS
- **ext4 noatime,writeback: 5432 TPS**
- zfs no tuning: 1776 TPS
- zfs atime off: 1835 TPS
- zfs atime off compression lz4: 4268 TPS
  Space used 986 MB
- **zfs atime off compression lz4,record 16k: 6288 TPS**
  Space used 2.2 GB

zfs module parameters

- zfs_prefetch_disable: "1"
- zfs_nocacheflush: "1"
- zfs_arc_max: "28294967296"
- zfs_arc_min: "28294967296"
- zfs_txg_timeout: "1"

Add a zil to the zpool

Example

```
 zpool status
  pool: pg_pool
 state: ONLINE
config:

        NAME                     STATE     READ WRITE CKSUM
        pg_pool                  ONLINE       0     0     0
          google-pg-zfs-hdd-0    ONLINE       0     0     0
        logs
          sde                    ONLINE       0     0     0
```

Example

```
dropping old tables...
creating tables...
generating data (client-side)...
100000 of 50000000 tuples (0%) done (elapsed 0.08 s, remaining 39.66 s)
.
.
50000000 of 50000000 tuples (100%) done (elapsed 55.49 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 75.91 s (drop tables 0.20 s, create tables 0.01 s, client-side generate
    55.69 s, vacuum 0.79 s, primary keys 19.23 s).
```

Space used roughly 2.2 GB

Example

```
starting vacuum...end. pgbench (16.2)
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 500
query mode: simple
number of clients: 50
number of threads: 1
maximum number of tries: 1
duration: 120 s
number of transactions actually processed: 748434
number of failed transactions: 0 (0.000%)
latency average = 7.977 ms
initial connection time = 705.825 ms
tps = 6268.124727 (without initial connection time)
```

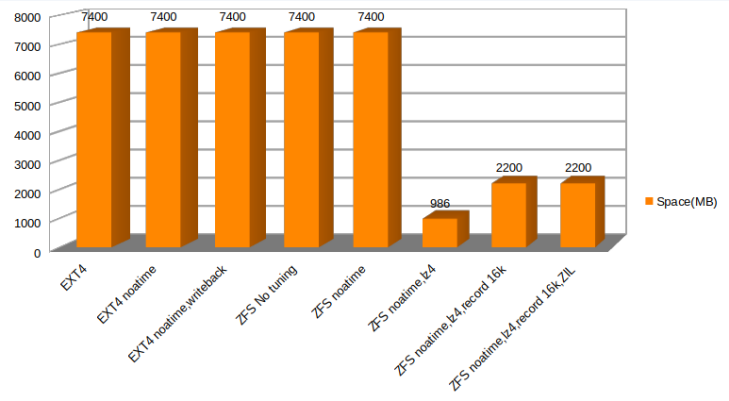
No improvement

KAMEDATA
DEVOPS AND DATA ENGINEERING

- ext4 no tuning: 4913 TPS
- ext4 noatime: 5371 TPS
- **ext4 noatime,writeback: 5432 TPS**
- zfs no tuning: 1776 TPS
- zfs atime off: 1835 TPS
- zfs atime off compression lz4: 4268 TPS
  Space used 986 MB
- zfs atime off compression lz4,record 16k: 6288 TPS
  Space used 2.2 GB
- **zfs module tweak, zil: 6268 TPS**
  Space used 2.2 GB

KAMEDATA
DEVOPS AND DATA ENGINEERING

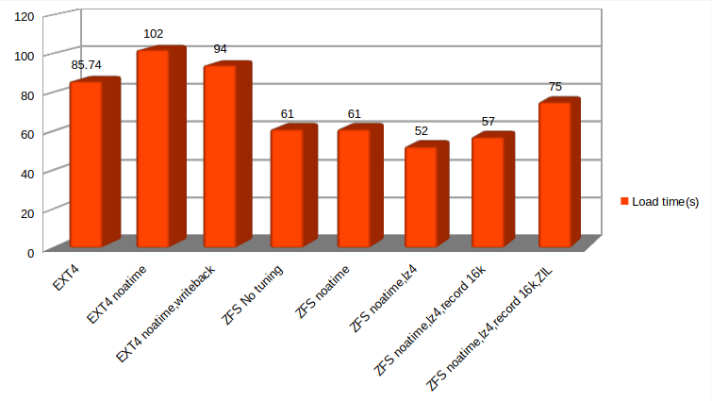
Space usage comparison (MB)



Smaller is better

KAMEDATA
DEVOPS AND DATA ENGINEERING

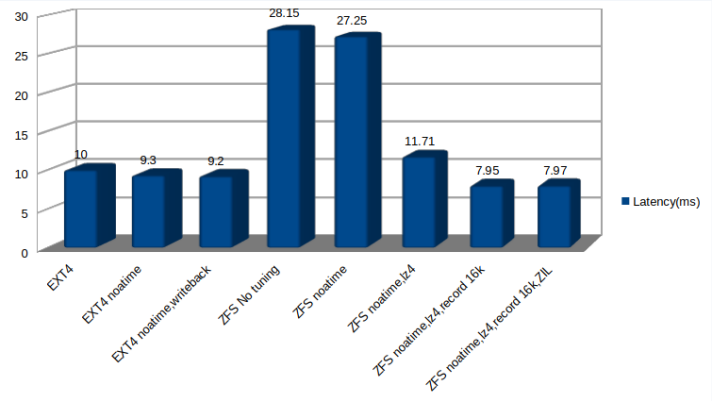
Load time comparison (Sec)



Smaller is better

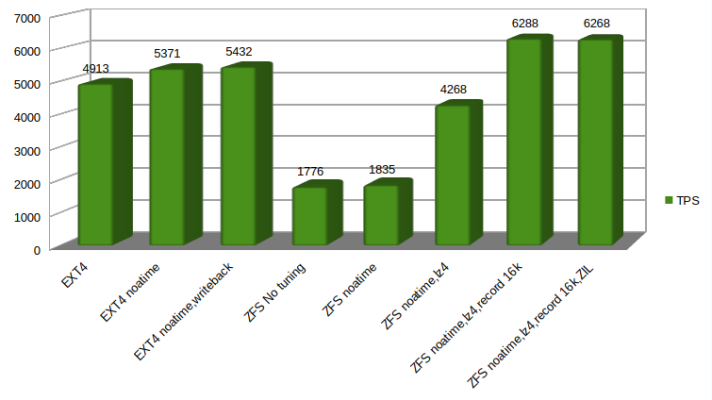
KAMEDATA
DEVOPS AND DATA ENGINEERING

Latency (ms)



Smaller is better

Transactions per second



Larger is better

Wrap up

- ZFS can achieve good performance despite the bad reputation
- Beware of any weird bug appearing...
- Yes, I mean this bug:
  https://www.theregister.com/2023/11/27/openzfs_2_2_0_data_corruption/
- Use the PostgreSQL block checksums
- Always use the right tool for the right job
  e.g. Don't use ZFS if you have a 20 MB database
- Always RTFM!
- And finally, remember that...

Y.M.M.V.

Translation: no airbags. we die as heroes

Thank you for listening!

Any questions?

Image sources

- LAPD badge - source wikicommons
- Base jumper - copyright Chris McNaught
- Disaster girl - source memegenerator
- Commodore 64 - source memecenter
- Deadpool- source memegenerator
- Sparta birds - source memestorage
- Mr Creosote - source Monty Python wiki
- Angry old man - source memegenerator

Vindicating ZFS with PostgreSQL: Unleashing the Power of Scalability

POSETTE 2024

Federico Campoli

June 2024