Latest evolution of Linux IO stack, explained for database people
Why this talk

- Linux is a most common OS for databases
- Fast IO is essential for many workloads
- DBAs often run into IO problems
- Most of the information on topic is written by kernel developers (for kernel developers) or is checklist-style
- Last years Linux IO stack (re)development is very fast
• How a generic database or PostgreSQL interacts with IO
• Linux IO as we used to understand it
• What is new?
Well, typical database

- DRAM
- Shared memory
  - WAL buffer
- Page cache
- Disks
  - WAL
  - Datafile

Database | Linux | User space | Kernel space
---------|-------|------------|-------------

It is easy, while read only

```
select foo from bar where foo=3
```

DataEGRET.com
Writes add complexity

update foo set bar=buzz

DRAM
shared_buffers
Page cache
Page
Dirty page
data/file
WAL
WAL buffer
worker

PostgreSQL
Linux
Page cache

Disks
data/file
WAL

Page
Dirty page
Key things about modern database workload

- Shared memory segment can be very large
- Keeping in-memory pages synchronized with disk generates huge IO
- WAL should be written fast and safe
- One and every layer of OS IO stack involved
What generates most of IO in case of PostgreSQL

- Keeping pages synchronized: checkpoints and other sync mechanisms
- Autovacuum can generate a lot of IO
- Cache refill
- Worker IO (Sorts and hashing, as well as worst-case fsyncs)
The main IO problem for databases for a long time was

- **How to maximize page throughput between memory and disks**
- Things involved:
  - Disks
  - Memory
  - CPU
  - IO Schedulers
  - Filesystems
  - Database itself
- IO problems for databases are not always only about disks
The main IO problem for databases for a long time was

- **How to maximize page throughput between memory and disks**
- Things involved:
  - Disks - because latency of this part was very significant
  - Memory
  - CPU
  - IO Schedulers
  - Filesystems
  - Database itself
- IO problems for databases are not always only about disks
• Maximizing IO performance through maximizing throughput is easy up to certain moment
• Minimizing latency of IO usually is tricky
• With large adoption of proper SSDs, hardware latency dropped dramatically
Because of high latency of rotating disks

- Database development was concentrated around maximization of throughput
- So did Linux kernel development
- Many rotating disks era IO optimization techniques are not that good for SSDs
IO stack (as it used to look like)

Database memory

Direct IO  VFS  EXT4  Page cache

BIO Layer

Request Layer  Elevator/IO Scheduler

Block device interface

Disks
IO stack (as it used to look like)

- **Database memory**
- **Direct IO**
- **VFS**
- **EXT4**
- **Page cache**
- **BIO Layer**
- **Request Layer**
- **Elevator/IO Scheduler**
- **Block device interface**
- **Disks**

Operates pages

Makes transition more efficient

Operates cylinders/sectors
• **Linus Elevator** - the only one in times of 2.4
• *merging* and *sorting* request queues
• Had **lots** of problems
- CFQ - universal, default one
- **deadline** - rotating disks
- *noop* or *none* - then disks throughput is so high, that it can not benefit from keen scheduling
  - PCIe SSDs
  - SAN disk arrays
Elevators: 3.13 and newer

- Effectiveness of **noop** clearly shows ineffectiveness of others, or ineffectiveness of smart sorting as an approach
- **blk-mq** scheduler was merged into 3.13 kernel
- Much better deals with parallelism of modern SSD - basically separate IO queue for each CPU
- The best option for good SSDs right now
- **blk-mq and NVMe driver is actually more than scheduler, but a system aimed to substitute whole request layer**
Old approach to elevators
New approach to elevators

CPU 1

sw queue

CPU 2

sw queue

CPU 3

sw queue

CPU 4

sw queue

hw queue

hw queue

Disks
Database memory

Direct IO

VFS

EXT4

Page cache

Block IO

BIO Layer

Kyber/BFQ IO schedulers

blk-mq

NVMe driver

Disks
Good diagram on Linux IO stack

- Regular updates
- Some things are difficult to draw, but it is a complex topic
Sets of standards, which helps to use modern SSDs more effectively
For Linux it is first of all NVMe driver (or subsystem)
Most common example of NVMe SSDs are PCIe NAND drives
With NVMe v.5 (currently 3 is ready for production) can work up to 32GB/sec
Are databases NVMe ready?
Latest development on new block layer

- IO polling
- New IO schedulers Kyber and BFQ (Kernel 4.12)
- IO tagging
- Direct IO improvements
Currently PostgreSQL supports DirectIO only for WAL, but it is unusable on practice

- Requires a lots of development
- Very OS specific
- Allows to use specific things, like O_ATOMIC
- PostgreSQL is the only database, which is not using Direct IO
Questions?

ik@dataegret.com