Linux tuning
for Database performance

Ilya Kosmodemiansky
ik@postgresql-consulting.com

PostgreSQL-Consulting.com

Bird-eye view

- Databases share common principles
- That means they can share common performance problems
- Database's workload is a complex workload, which utilizes many OS mechanisms
- We need to understand those mechanisms and tune OS well to achieve maximum performance
- My primary focus is PostgreSQL, but I used to work with Oracle, DB2, MySQL as well
- So I will try to talk about common practices, adding details about exact database when necessary

What is not about

- Monitoring. A very huge and important topic, we will only touch it
- Database complex performance tuning. We will do it only in scope of OS and hardware interactions

Why Linux?

- Most impressive progress about implementing features which affect database performance
- Many of them were sponsored or implemented by database vendors
- Wide adoption

The modern linux kernel

- About 1000 sysctl parameters (plus non-sysctl settings, such as mount options)
- A default linux configuration is for a default workload
- It is not possible to benefit from the modern kernel's advantages without wise tuning

Tuning targets in Linux

- CPU
- Memory
- Storage
- Other

Database specifics is

- Tuning single target can have a very small effect
- Whole operating system can be a tuning target
- Sometimes it can be very complex, we need to decompose the task
- Generally, it is a bad idea to share a server between database and something else (except pgbouncer)

Database performance based on 3 things

- A proper hardware
- OS tuning
- Database tuning

Thank you, Captain Obvious!

What is a proper hardware?

- Big data likes powerful hardware
- If you can not afford powerful hardware, try small data (I am serious)
- Well-balanced hardware: it is difficult to utilize 512Gb of RAM, running a database on single SATA-disk
- You need to know it

Choosing CPUs

- Database workload can be CPU-consuming
- In case of Postgres too:
  - It uses processes - workers, backends
  - Besides workers, there are a bunch of house-keeping processes: logwriter, several sorts of autovacuum, checkpointer bgwriter, replication-related processes
  - many things are pretty straight-forward
  - plpgsql is an interpretable language
- Most likely you will need many cores/CPUs rather than very fast CPUs

Choosing RAM

- As many as you can afford
- A good idea is, to have enough room in a server to expand the amount of memory on demand, while your system is growing
- Take in mind, that it can be not an easy task to bye additional RAM of that particular type in 5 years
- Once again: a database hardly can benefit from 512Gb of top-notch RAM on two CPU and one SATA disk.

Choosing disk subsystem

- Basically you have 3 options:
  - SSD
  - SAS
  - SATA
- Generally, today SSD is the best solution, but there are issues
- SATA + heavy workload = no way

SAS

- Server-class, 15K, 2,5" for faster seek
- With SAS'es you definitely need a RAID
- RAID, managed by a proper RAID controller

RAID controller

- Software and hardware
- Hardware RAID controller must not be a cheap one. Chip on controller has a price, so cheap means the controller uses servers CPU - even Software RAID will be better
- Historically, we recommend LSI/Perc to customers, but there are more good controllers
- Some controllers from for example HP or Areca have issues we had run into
- For using with databases a controller must have cache and a battery **for rotating disks at least**

SSD

- **Desktop-class SSD can be worser than SAS**
- Server class: Intel 3700, 3600-series
- You can use server class (or datacenter family, how Intel calls them) SSD with software RAID or bypass controller's cache
- Because server-class SSDs usually have their own cache

Memory is faster than disks

That is a common knowledge

Shared memory is the first step for many people

- shared_buffers in Postgres, innodb_buffer_pool_size in MySQL
- Rule of thumb: 25% of available DRAM
- Oracle's 40% do not work: that will be a worst case of double caching for Postgres
- MySQL now can O_DIRECT, but there are some issues as far as I know
- Modern server can easily have 512Gb of DRAM... and here the fun begins

The fun

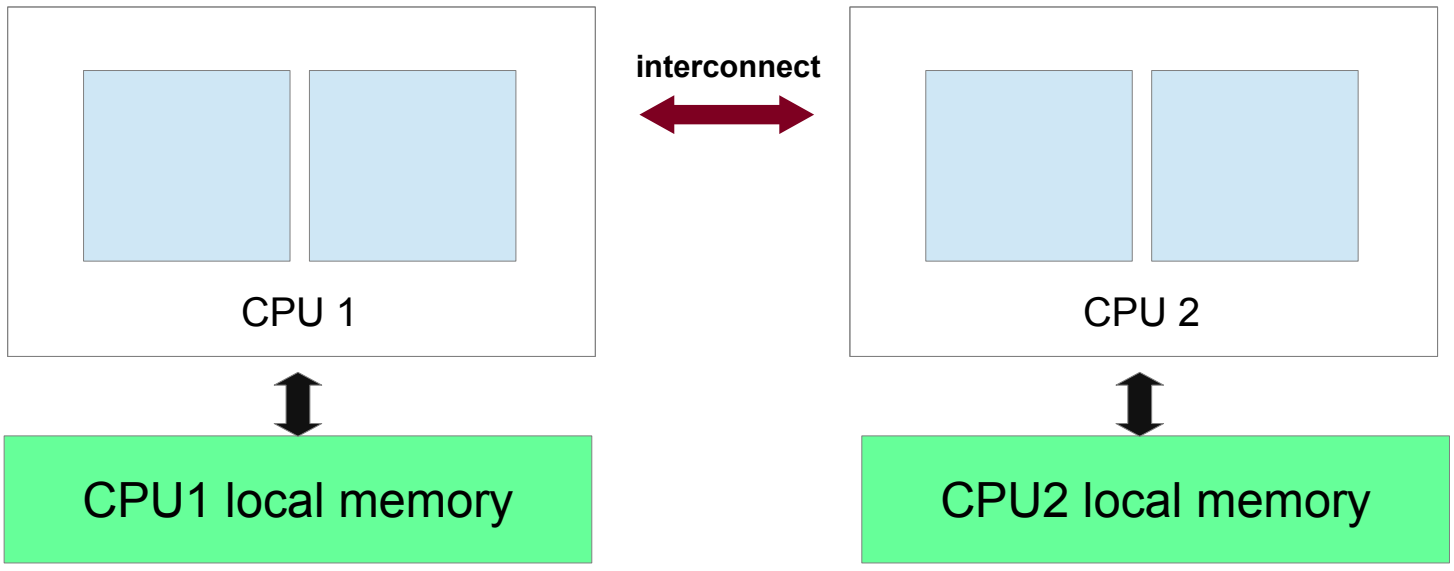
- NUMA
- Huge pages
- Swap

NUMA

- Non Uniform Memory Access
- CPUs have their own memory, CPU + memory nodes connected via NUMA interconnect
- CPU uses its own memory, then accesses remaining memory by interconnect (*numactl – – hardware shows distances*)
- If node interleaving disabled, CPU tries to use its local memory (for page cache for example;-))

PERCONA LIVE

NUMA

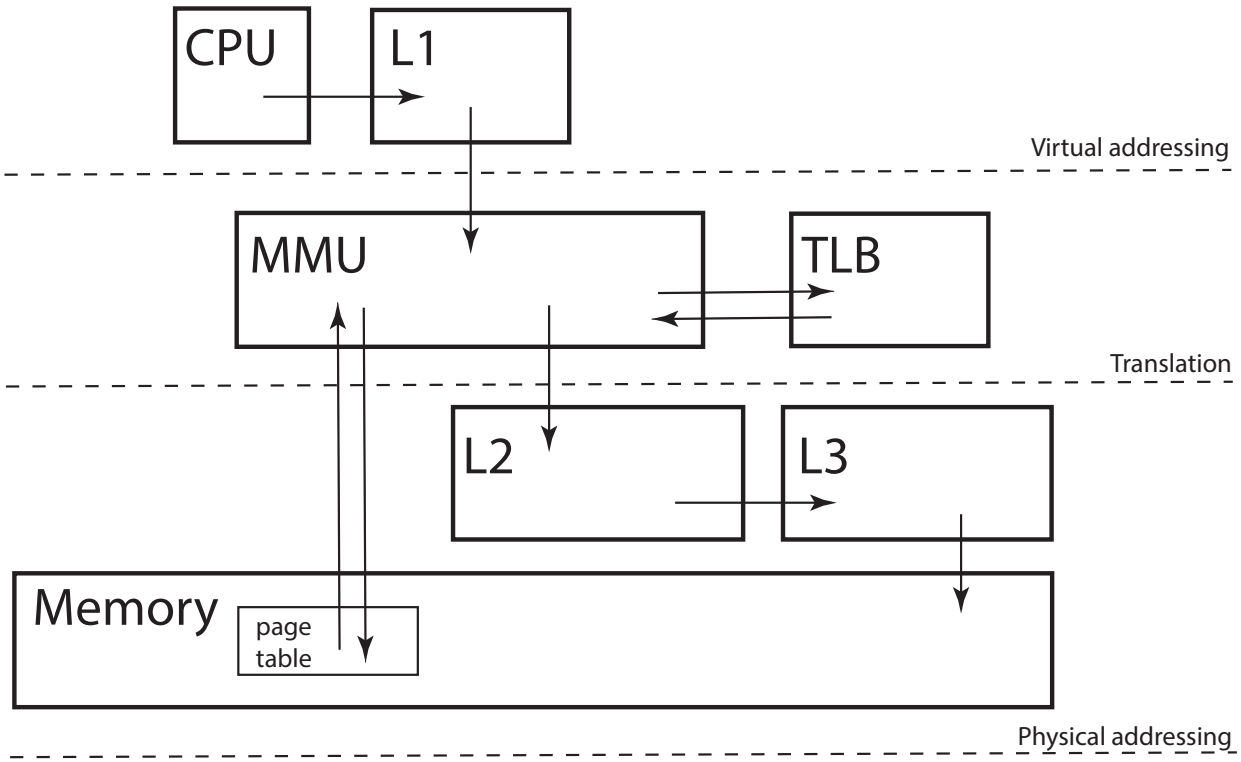


NUMA

Which NUMA configuration is better for PostgreSQL

- Switch NUMA off at low level (better say make OS unaware of NUMA)
  - Enable memory interleaving in BIOS
  - numa $\rightarrow$ off at kernel boot
- Another approach
  - *vm.zone_reclaim_mode* = 0
  - *numactl* $--$*interleave* = *all* */etc/init.d/postgresql start*
  - kernel.numa_balancing = 0

That was true for MySQL as well for years, and may be still it is, because innodb_numa_interleave was introduced not a long time ago


CPU
L1
Virtual addressing
MMU
TLB
Translation
L2
L3
Memory
page
table
Physical addressing

Huge pages

What goes on

- By default OS allocates memory by 4kB chunks
- OS translates physical addresses into virtual addresses and cache the result in TLB
- $\frac{1Gb}{4kB} = 262144$ - huge TLB overhead and cache misses
- Better to allocate memory in larger chunks

Huge pages

How can PostgreSQL benefit from huge pages?

- Enable pages in kernel
- *vm.nr_hugepages* = 3170 via sysctl
- Before 9.2 - libhugetlbfs library
- 9.3 - no way
- 9.4+ *huge_pages* = *try*|*on*|*off* (postgresql.conf)
- Works on Linux, not for example on FreeBSD
- Disable Transparent huge pages - it is easy to run into problems with many databases

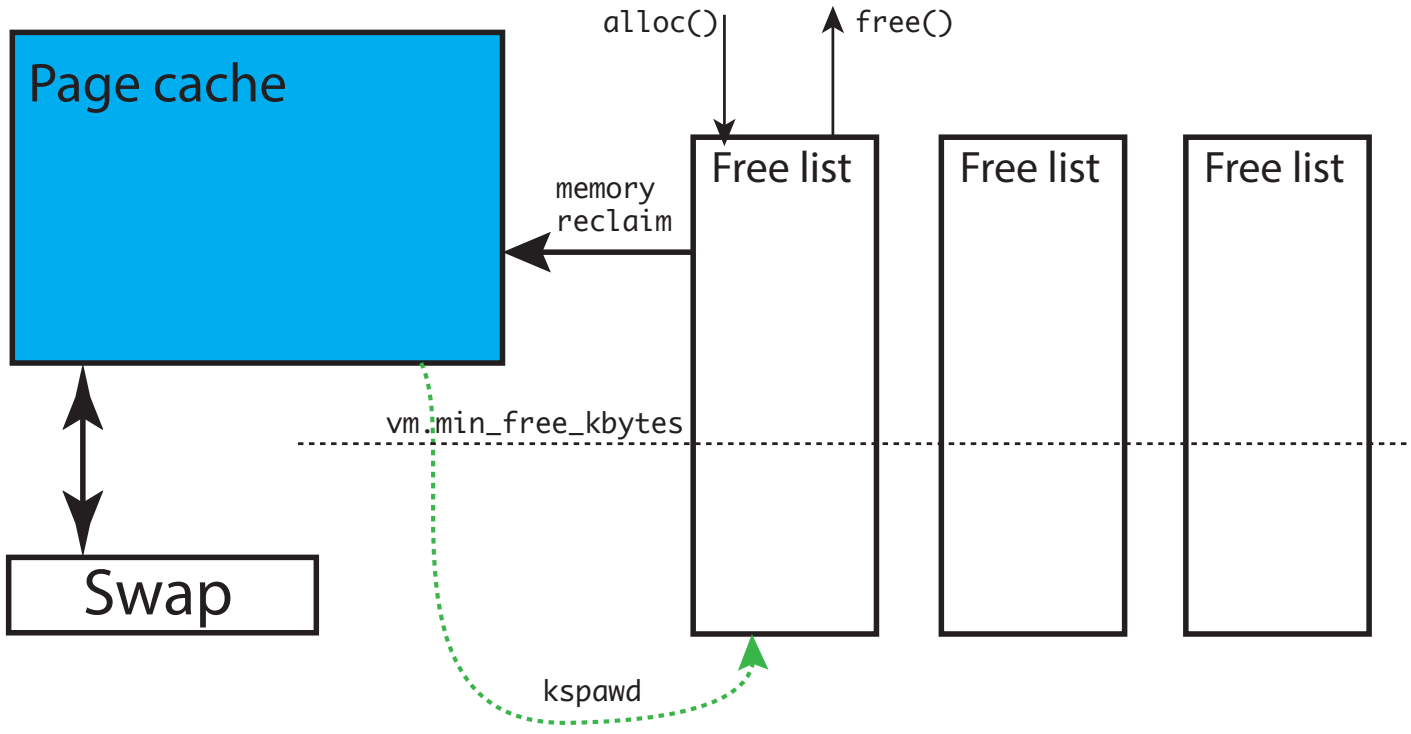
MySQL can use –large-pages to benefit from huge pages

Swap

- There are enough memory, but swap is used

PERCONA LIVE

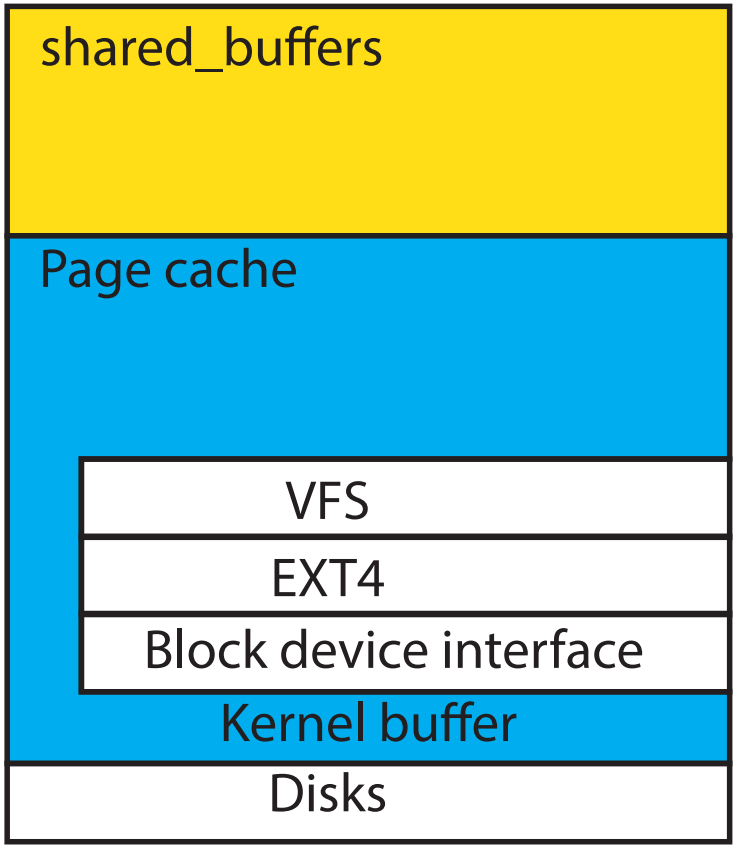
How Linux work with memory - 2



Swap

- vm.swappiness controls a degree, how many memory will be swapped or paged out (from 0 to 100)
- vm.swappiness = 60 by default
- 0 is a bad idea because of OOM-killer
- Low figures, such as 1-10 are better, for databases 1 can be a recommended value

PostgreSQL-Linux IO stack (simplified)



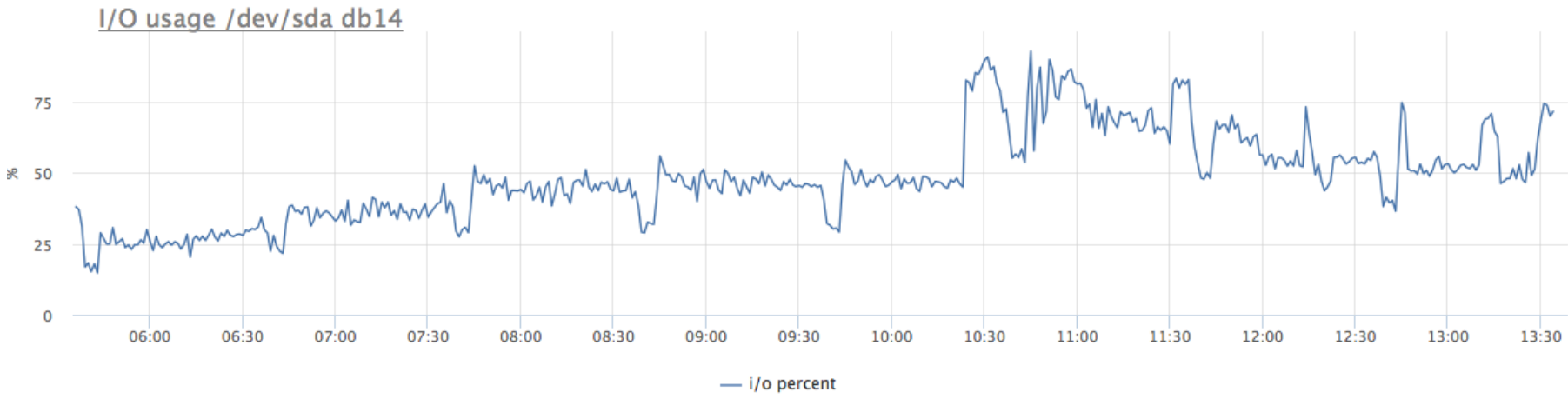
More effective flushing pages to disk

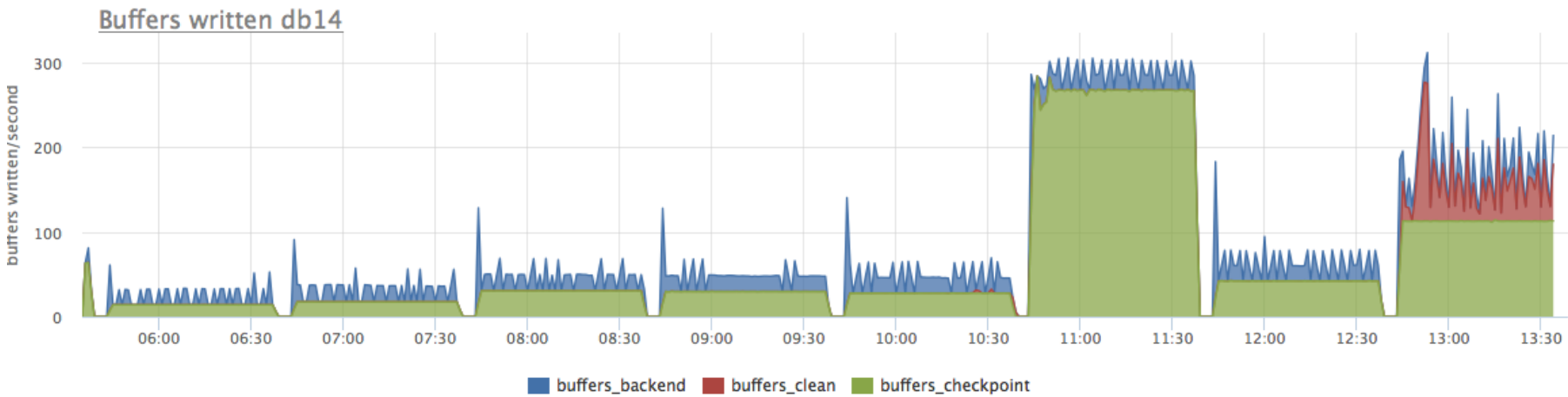
Symptoms that something goes wrong

- Checkpoint spikes

PERCONA LIVE

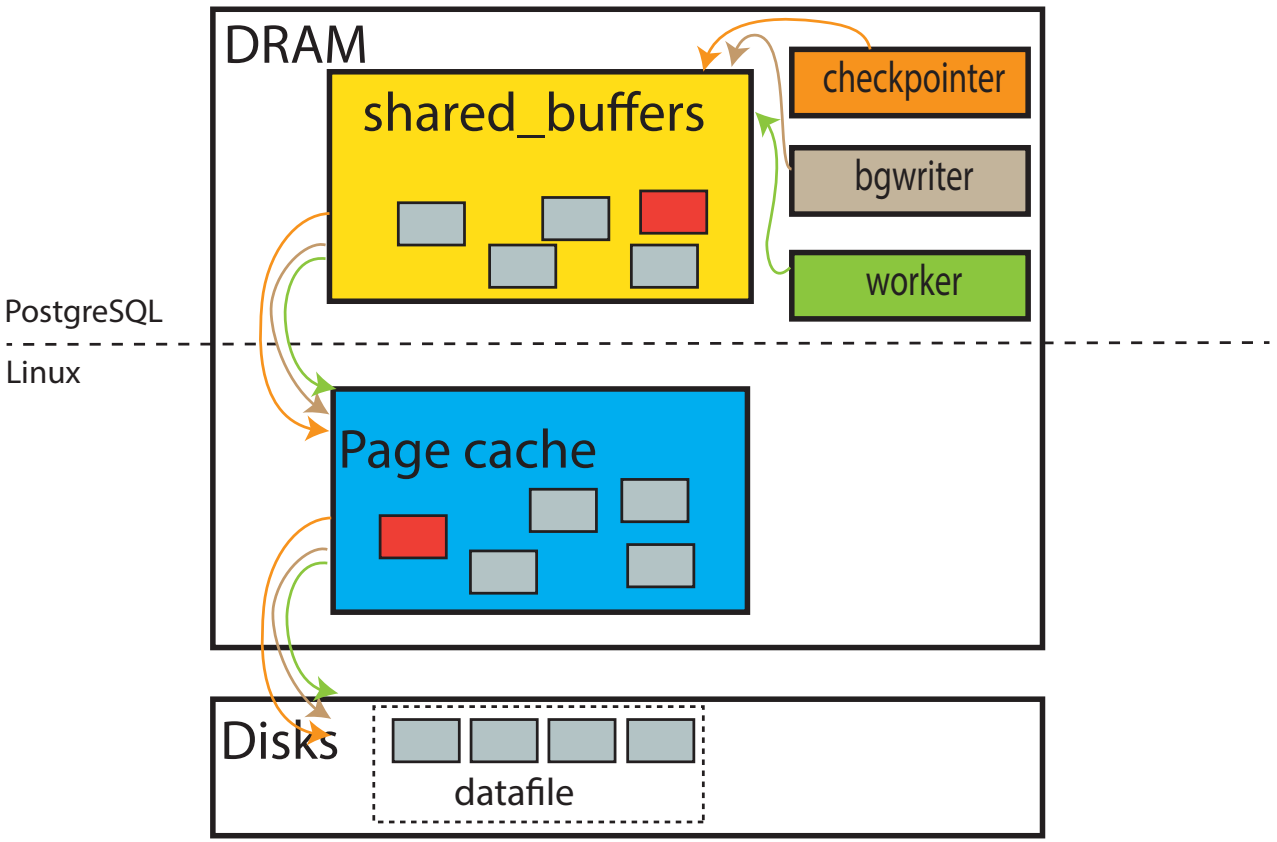
Checkpoint spikes





PERCONA LIVE

Three ways of syncing dirty pages in PostgreSQL



What is a spike

- Most likely, fsync from checkpoint overwhelms disks
- Bgwriter can not flush pages effectively
- Workers need more buffer cache, so they need to sync dirty pages by themselves
- Last kind if sync is an emergency measure and is very ineffective

What to do

- Find a proper hardware and set it up correctly
- Tune filesystem
- Tune VM
- Tune checkpoints and bgwriter in postgresql.conf

Rotating disks

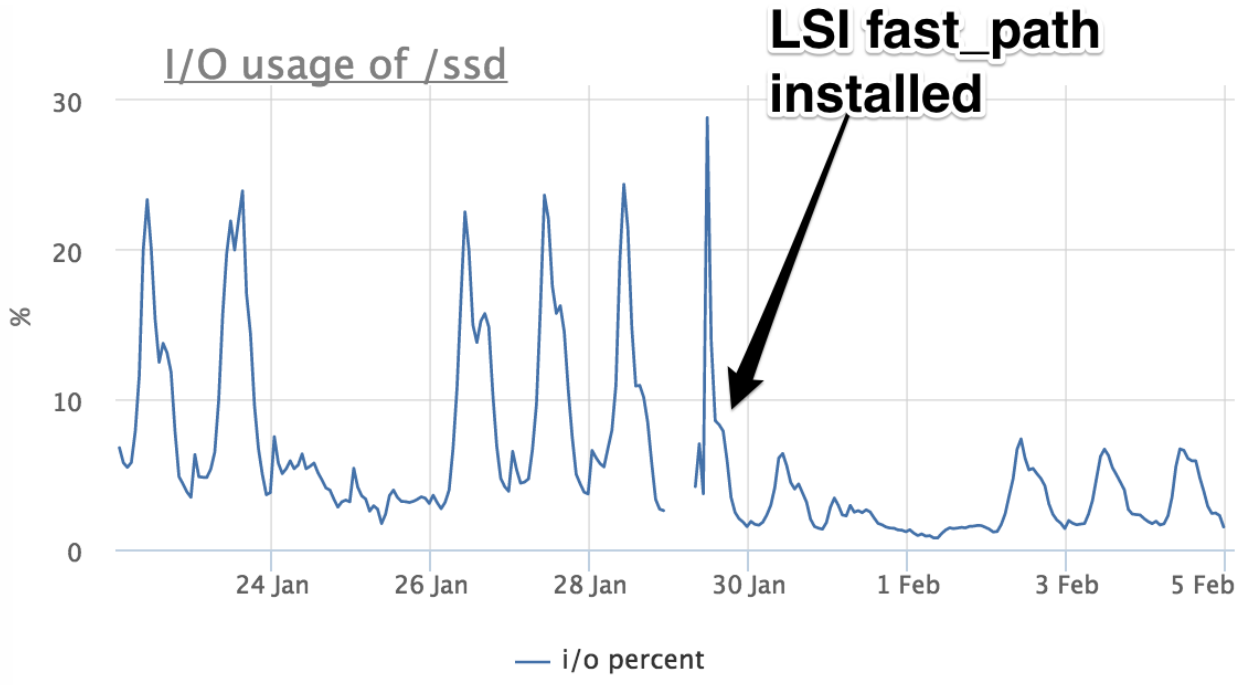
- Battery should be in a good condition
- cache mode $\rightarrow$ write back
- io mode $\rightarrow$ direct
- Disk Write Cache Mode $\rightarrow$ disabled

SSDs

- It is generally better to use disk cache, than a controller cache
- **only if you have server-class SSDs** with capacitor
- LSI has fast_path to bypass controllers cache
- New developments such as Seagate's new 10GB/s SSD with 16-lane PCIe slots

PERCONA LIVE

LSI fast path

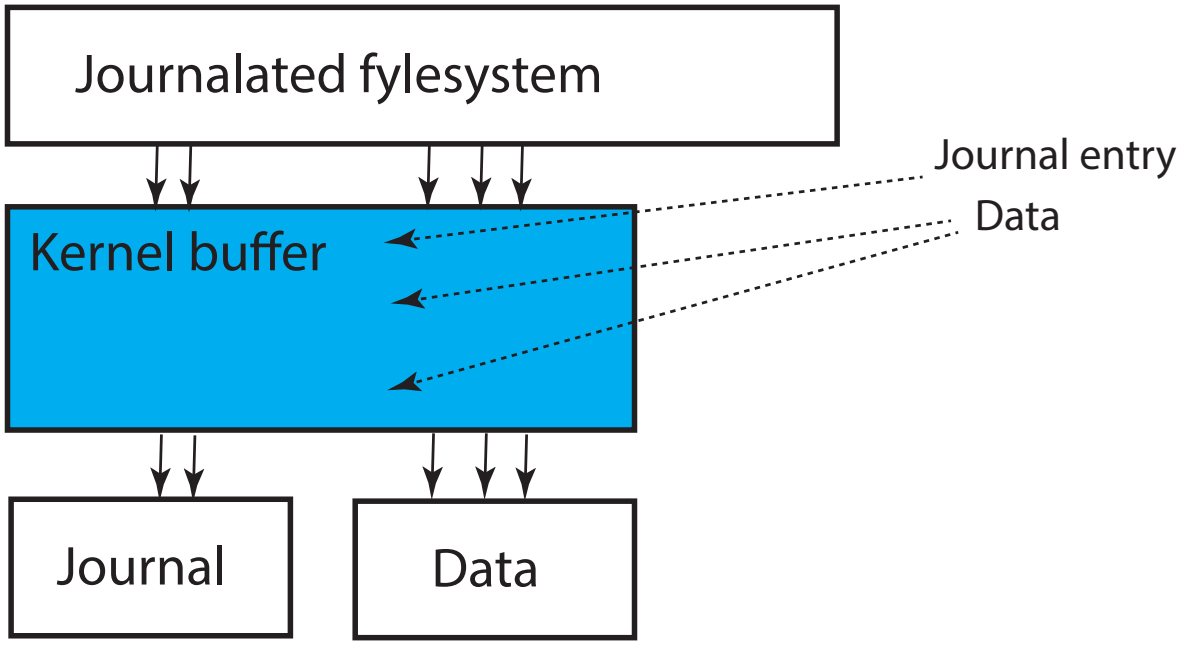


Filesystem tuning

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

A good talk by Tomas Vondra - http://www.slideshare.net/fuzzycz/postgresql-on-ext4-xfs-btrfs-and-zfs

Barrier


Journalated fylesystem
Journal entry
Data
Kernel buffer
Journal
Data

VM tuning

- By default vm.dirty_ratio = 20, vm.dirty_background_ratio = 10
- Nothing happens until kernel buffer is 10% full of dirty pages
- From 10% to 20% - background flushing
- From 20% IO seriously hurt until flush finishes its job
- This is almost crazy if your shared_buffers setting is 32Gb/64Gb or more with any cache on RAID-controller or SSD

VM tuning

What is better for PostgreSQL?

- vm.dirty_background_bytes = 67108864, vm.dirty_bytes = 536870912 (for RAID with 512MB cache on board) looks more reasonable
- Hardware settings and checkpoint settings in postgresql.conf must be appropriate
- Things will improve in 9.6 with backend_flush_after
- For those settings please check https://www.youtube.com/watch?v=Lbx-JVcGIFo

What else

- Scheduler tuning
- Power saving

Scheduler tuning

- sysctl kernel.sched_migration_cost_ns supposed to be reasonably high
- sysctl kernel.sched_autogroup_enabled = 0
- A good explanation *http://www.postgresql.org/message-id/50E4AAB1.9040902@optionshouse.com*
- You need a relatively new kernel

Example

```
$ pgbench -S -c 8 -T 30 -U postgres pgbench transaction type: SELECT only
scaling factor: 30 duration: 30 s
number of clients: 8 number of threads: 1

sched_migration_cost_ns = 50000, sched_autogroup_enabled = 1
- tps: 22621, 22692, 22502

sched_migration_cost_ns = 500000, sched_autogroup_enabled = 0
- tps: 23689, 23930, 23657
```

tests by Alexey Lesovsky

Power saving policy

- acpi_cpufreq and intel_pstate drivers
- scaling_governor: performance, ondemand, conservative, powersave, userspace
- acpi_cpufreq + performance can be dramatically faster than acpi_cpufreq + ondemand
- intel_pstate + powersave

Thanks

to my collegues Alexey Lesovsky and Max Boguk for a lot of research on this topic

Questions?

ik@postgresql-consulting.com