Linux tuning to improve PostgreSQL performance

Ilya Kosmodemiansky
ik@postgresql-consulting.com

PostgreSQL-Consulting.com

The modern linux kernel

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

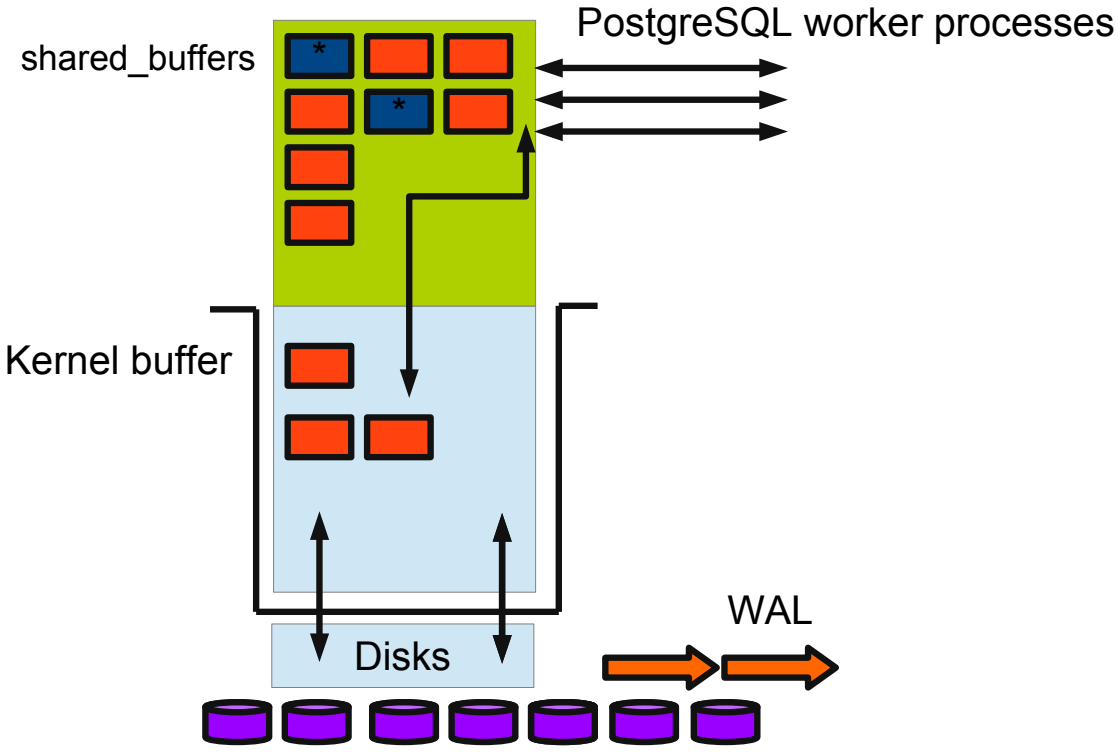
Tuning targets in Linux

- CPU
- Memory
- Storage
- Other

PostgreSQL specifics

- Hungry for resources (like any other database)
- Tuning single target can have a very small effect
- We need to maximize throughput

Throughput approach



How to make pages travel faster from disk to memory

- More effective work with memory
- More effective flushing pages to disk
- A proper hardware, of course

More effective work with memory

- NUMA
- Huge pages
- Swap

NUMA

Symptoms that something goes wrong

- Some CPU cores are overloaded without any obvious reason

NUMA

What goes on

- 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;-))

NUMA

Which NUMA configuration is better for PostgreSQL

- Switch NUMA off at low level
  - Enable memory interleaving in BIOS
  - numa → off at kernel boot
- Another approach
  - *vm.zone_reclaim_mode* = 0
  - *numactl −−interleave = all /etc/init.d/postgresql start*
  - kernel.numa_balancing = 0

Blog post from Robert Haas:
http://rhaas.blogspot.co.at/2014/06/linux-disables-vmzonereclaimmode-by.html

Huge pages

Symptoms that something goes wrong

- You have a lot of RAM and you shared_buffers settings is 32Gb/64Gb or more
- That means that you definitely have an overhead if not using huge pages

Huge pages

What goes on

- By default OS allocates memory by 4kB chunk
- OS translates physical addresses into virtual addresses and cache the result in Translation Lookaside Buffer (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
- Disable Transparent huge pages - PostgreSQL can not benefit from them

Swap

Symptoms that something goes wrong

- There are enough memory, but swap is used

Swap

What goes on

- It happens when there are a lot of RAM on server

Swap

What is better for PostgreSQL?

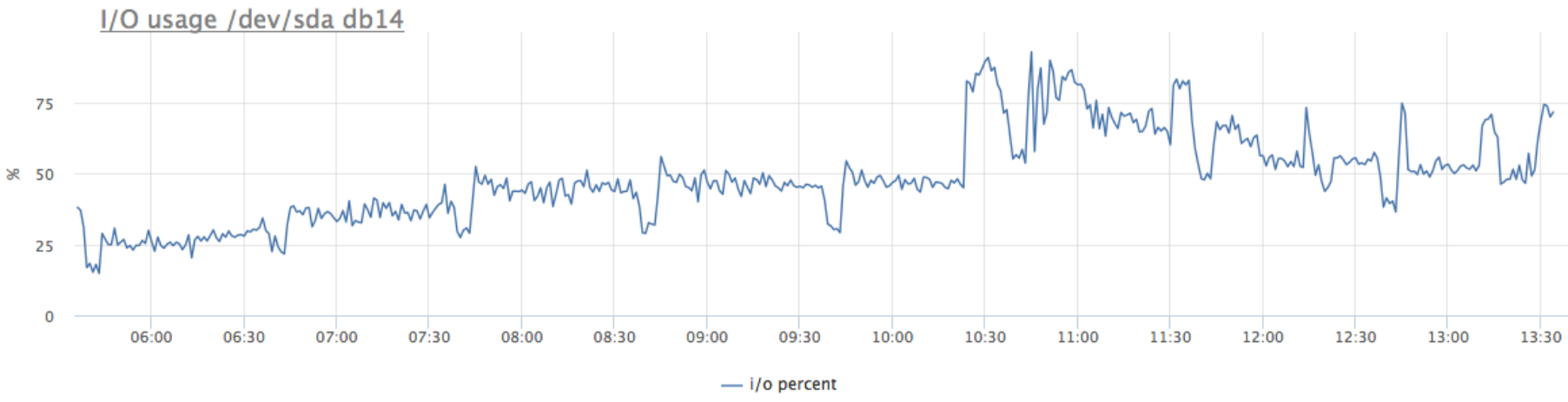
- vm.swappiness = 1 or 0
- OOM-killer
- 0 is not a good idea for modern kernels

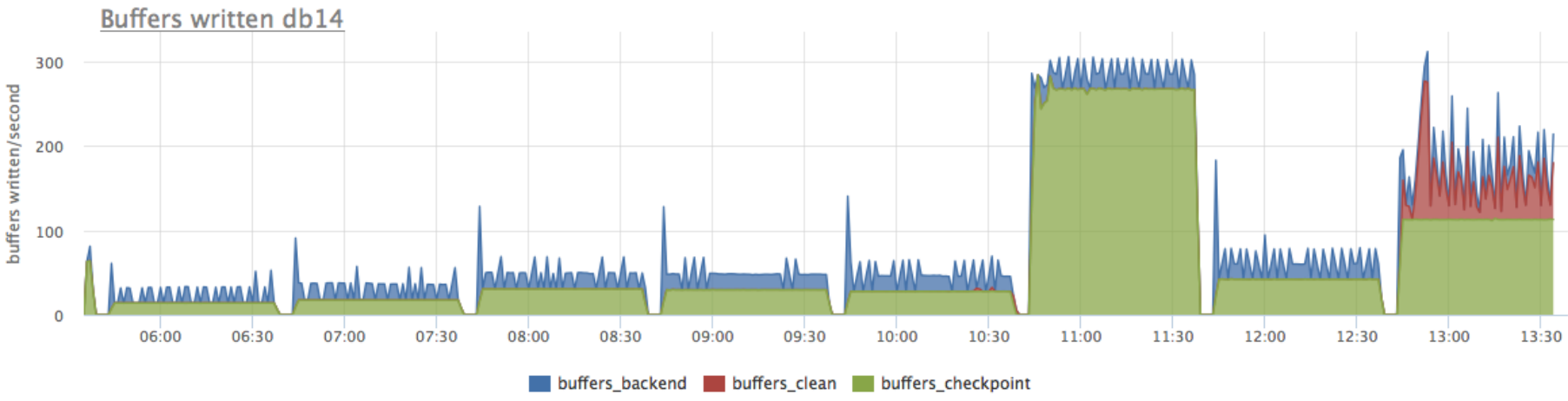
More effective flushing pages to disk

Symptoms that something goes wrong

- Checkpoint spikes

More effective flushing pages to disk





More effective flushing pages to disk

What goes on

- 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 effectively stops until pdflush/flushd/kdflush 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

More effective flushing pages to disk

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
- See my talk about PostgreSQL disc performance for details (*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