



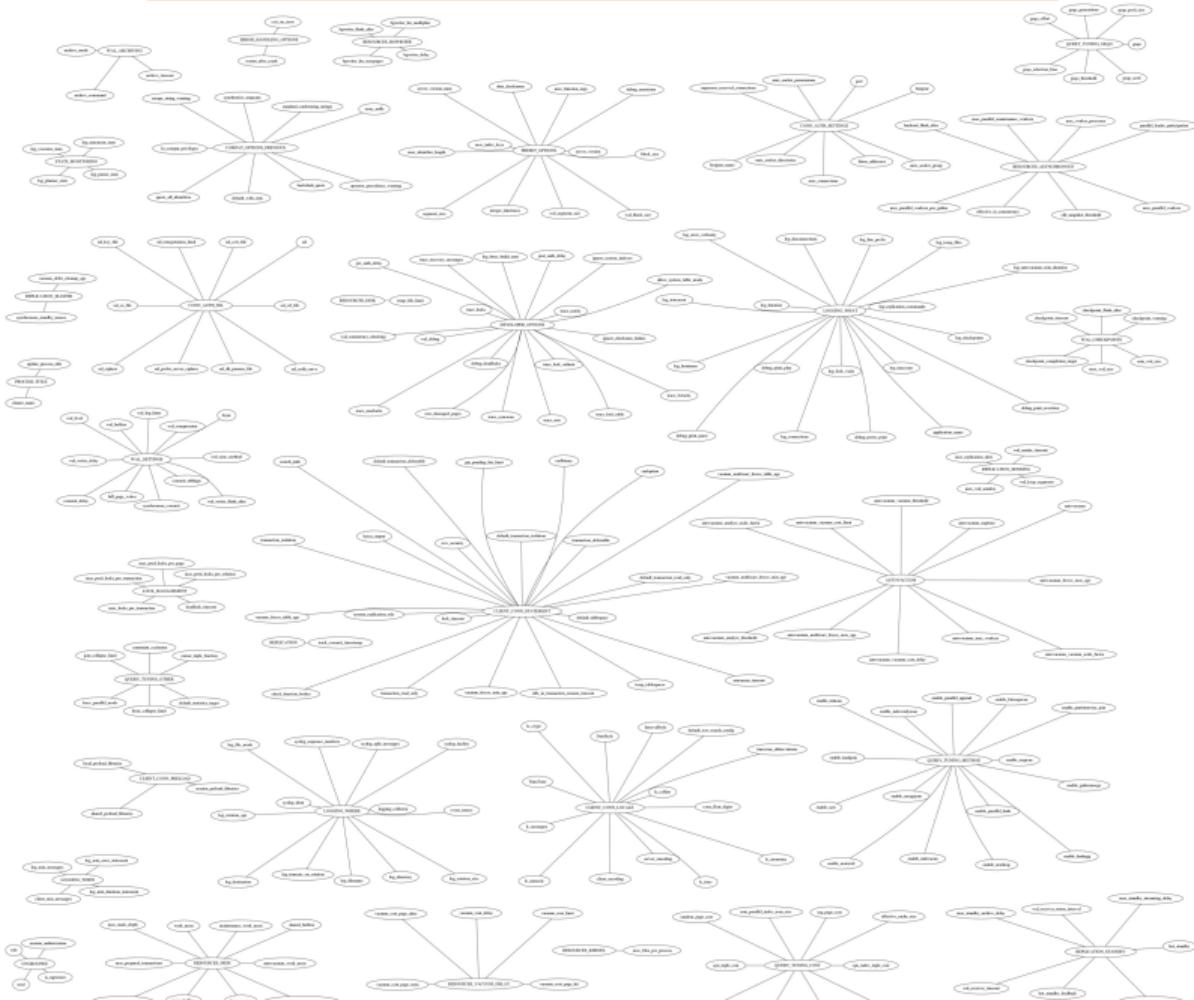
POSTGRESQL LINUX KERNEL

FRIENDSHIP

DMITRY DOLGOV

20-03-2018





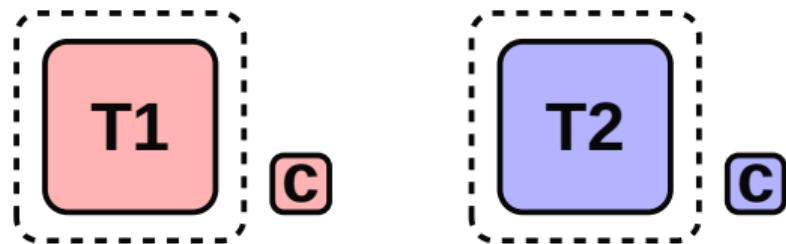
Execution scheduling

```
# Experiment 1
transaction type: pg_long.sql
latency average = 1312.903 ms

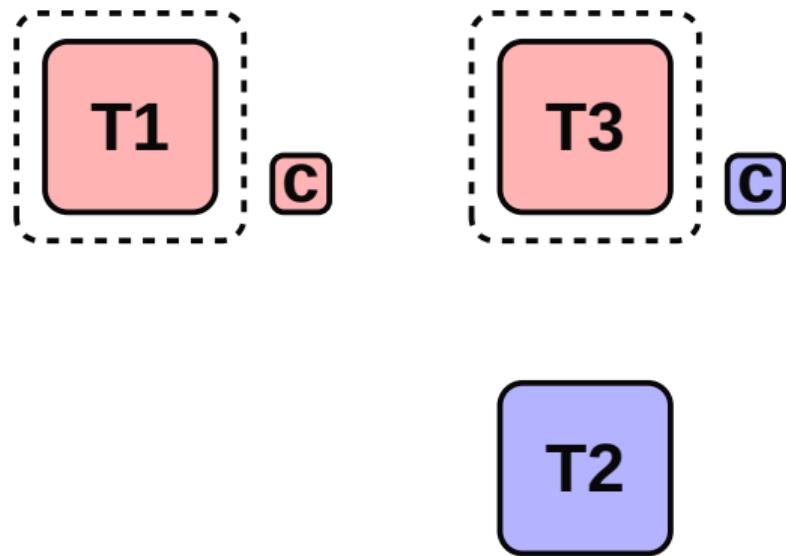
# Experiment 2
SQL script 1: pg_long.sql
- weight: 1 (targets 50.0% of total)
- latency average = 1426.928 ms

SQL script 2: pg_short.sql
- weight: 1 (targets 50.0% of total)
- latency average = 303.092 ms
```

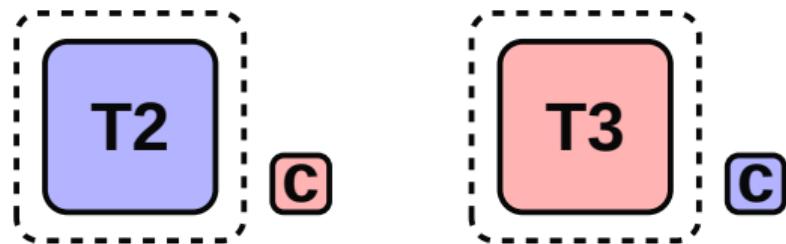
Scheduling



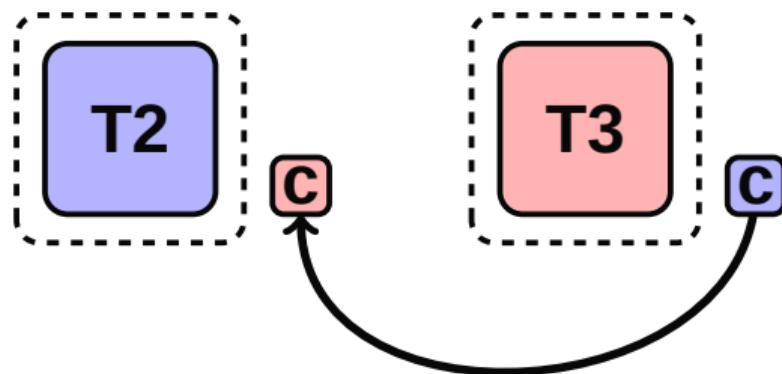
Scheduling



Scheduling



Scheduling



```
# Experiment 1  
12,396,382,649      cache-misses # 28.562%  
2,750                cpu-migrations  
  
# Experiment 2  
20,665,817,234      cache-misses # 28.533%  
10,460               cpu-migrations
```

time	cpu	01234	task name [tid/pid]	wait time (msec)	sch delay (msec)	run time (msec)	
4227.834476	[0003]	s	postgres[12935]	0.000	0.000	0.000	
4227.834895	[0003]	s	postgres[12935]	0.000	0.000	0.418	
4227.835478	[0003]	s	postgres[25080]	0.000	0.040	0.583	
4227.836485	[0003]	s	postgres[25080]	0.000	0.000	1.007	
4227.837482	[0003]	s	postgres[25080]	0.000	0.000	0.996	
4227.837784	[0003]	s	postgres[25080]	0.000	0.000	0.302	
4227.837989	[0003]	m	postgres[25080]				migrated: :25077 cpu 1 => 3
4227.837993	[0003]	s	postgres[25080]	0.000	0.000	0.208	
4227.848487	[0003]	s	postgres[25080]	0.000	0.000	10.493	
4227.848991	[0003]	s	postgres[25080]	0.000	0.000	0.504	
4227.849487	[0003]	s	postgres[25080]	0.000	0.000	0.495	
4227.849748	[0003]	s	postgres[25080]	0.000	0.000	0.260	
4227.849912	[0003]	s	postgres[25080]	0.000	0.000	0.164	
4227.851477	[0001]	s	postgres[25082]	0.000	0.000	0.000	
4227.851481	[0002]	s	postgres[25080]	0.000	0.000	0.000	
4227.851778	[0003]	s	postgres[12935]	15.017	0.000	1.866	
4227.852259	[0003]	m	postgres[12935]				migrated: postgres[25083] cpu 1 => 3
4227.852263	[0003]	s	postgres[12935]	0.000	0.000	0.484	
4227.852477	[0003]	s	postgres[25083]	0.000	0.058	0.214	
4227.852478	[0001]	s	postgres[25082]	0.000	0.000	1.001	
4227.852614	[0002]	s	postgres[12935]	0.000	0.000	1.133	

Tunables

Tunables

→ /proc/sys/kernel/sched_migration_cost_ns

Tunables

→ /proc/sys/kernel/sched_migration_cost_ns
→ 500000

Tunables

- /proc/sys/kernel/sched_migration_cost_ns
- 500000
- /proc/sys/kernel/sched_wakeup_granularity_ns

Tunables

- /proc/sys/kernel/sched_migration_cost_ns
- 500000
- /proc/sys/kernel/sched_wakeup_granularity_ns
- 2000000

Tunables

- /proc/sys/kernel/sched_migration_cost_ns
- 500000
- /proc/sys/kernel/sched_wakeup_granularity_ns
- 2000000
- /proc/sys/kernel/sched_min_granularity_ns

Tunables

- /proc/sys/kernel/sched_migration_cost_ns
- 500000
- /proc/sys/kernel/sched_wakeup_granularity_ns
- 2000000
- /proc/sys/kernel/sched_min_granularity_ns
- 1500000

Tunables

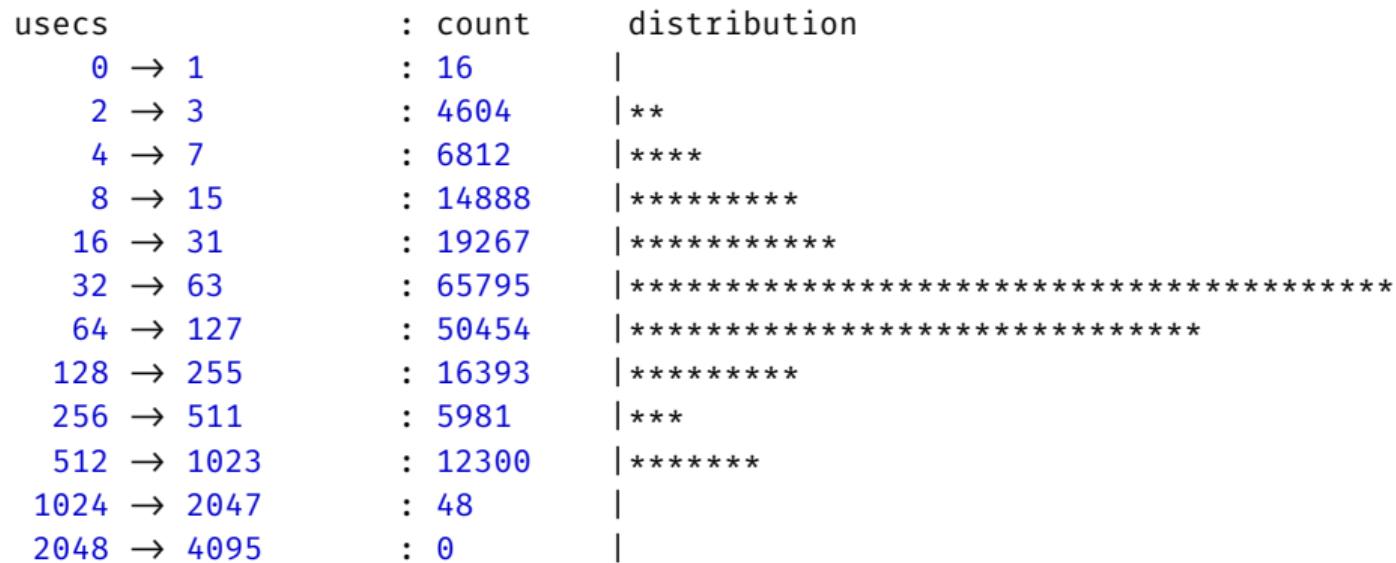
- /proc/sys/kernel/sched_migration_cost_ns
- 500000
- /proc/sys/kernel/sched_wakeup_granularity_ns
- 2000000
- /proc/sys/kernel/sched_min_granularity_ns
- 1500000
- /proc/sys/kernel/sched_latency_ns

Tunables

- /proc/sys/kernel/sched_migration_cost_ns
- 500000
- /proc/sys/kernel/sched_wakeup_granularity_ns
- 2000000
- /proc/sys/kernel/sched_min_granularity_ns
- 1500000
- /proc/sys/kernel/sched_latency_ns
- 1200000

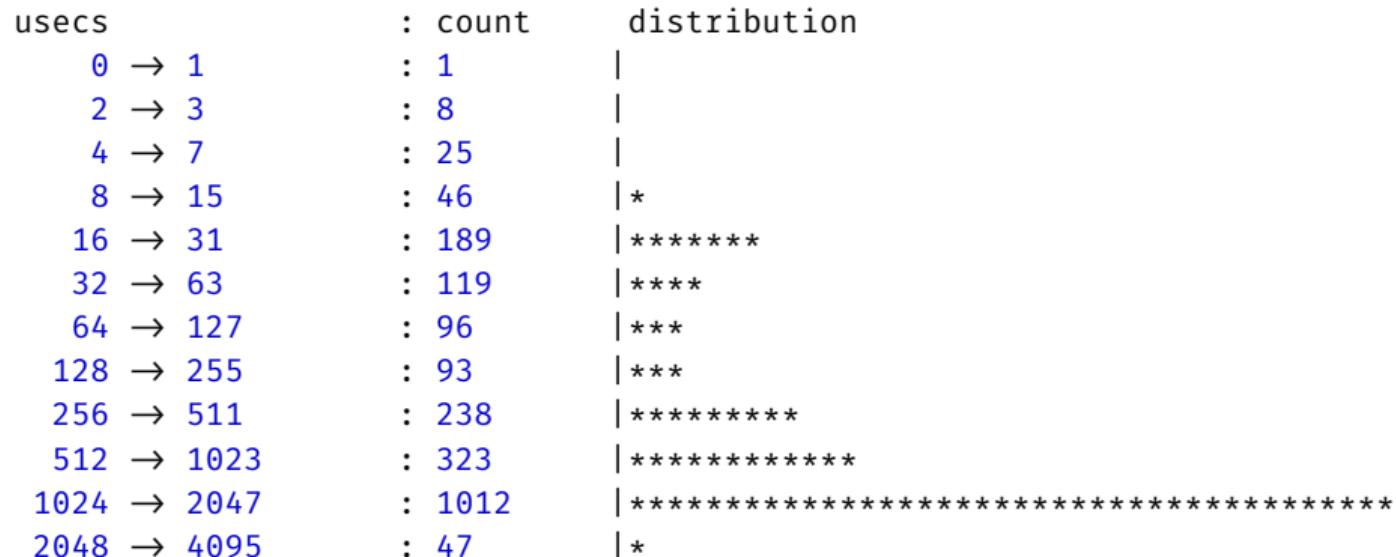
pgbench and pg_dump

```
real    1m38.990s  
user    1m9.127s  
sys     0m2.066s
```

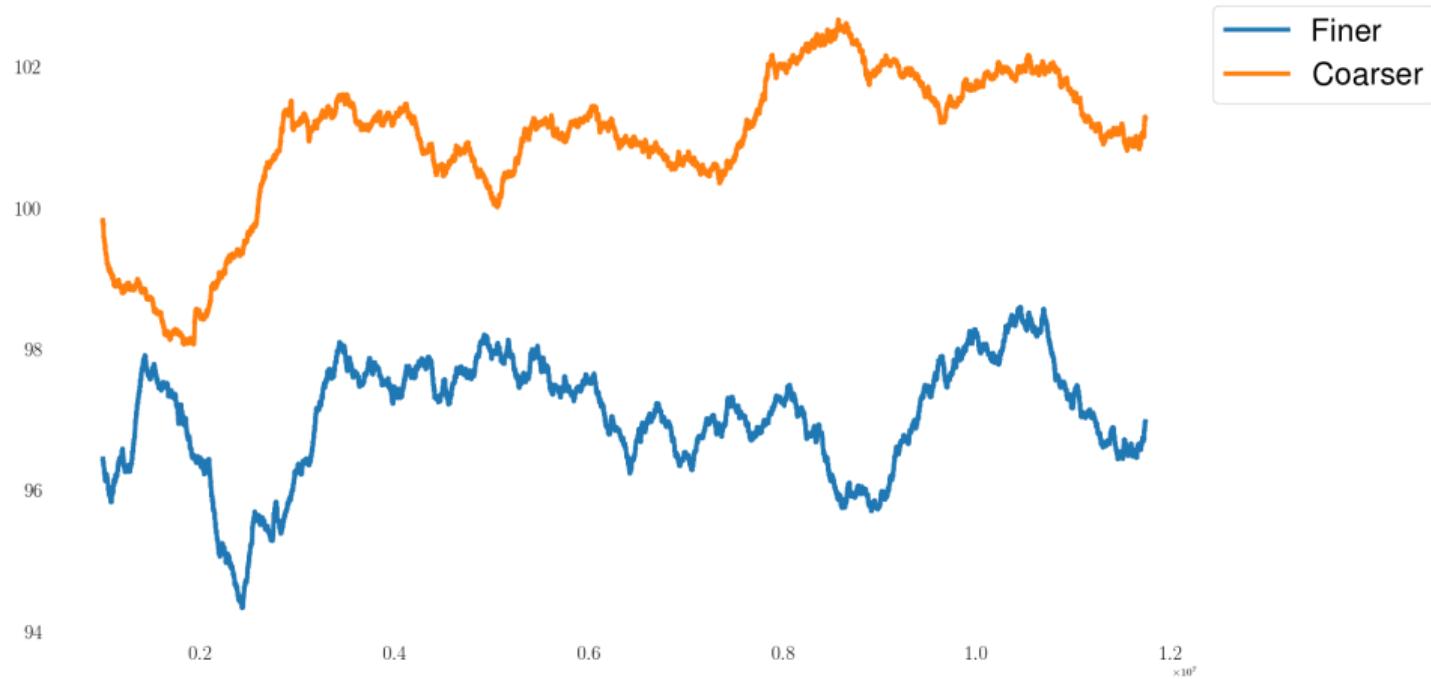


pgbench and pg_dump

```
real    1m32.030s
user    1m8.559s
sys     0m1.641s
```



Wakeup granularity, microsec



HyperThreading

Intel® 64 and IA-32 Architectures Optimization Reference Manual

HyperThreading

→ Less deviation for latency

Intel® 64 and IA-32 Architectures Optimization Reference Manual

HyperThreading

- Less deviation for latency
- Share execution state and cache

Intel® 64 and IA-32 Architectures Optimization Reference Manual

HyperThreading

- Less deviation for latency
- Share execution state and cache
- Spin locks have significant impact

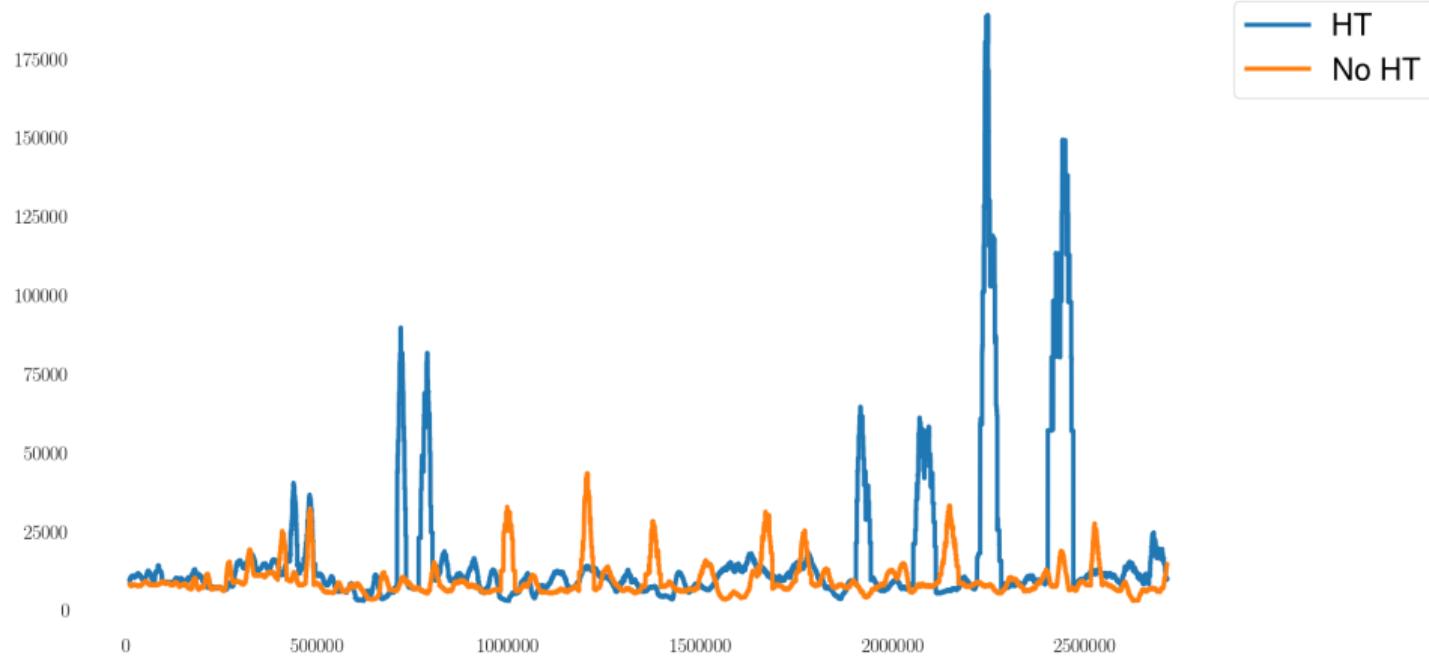
Intel® 64 and IA-32 Architectures Optimization Reference Manual

HyperThreading

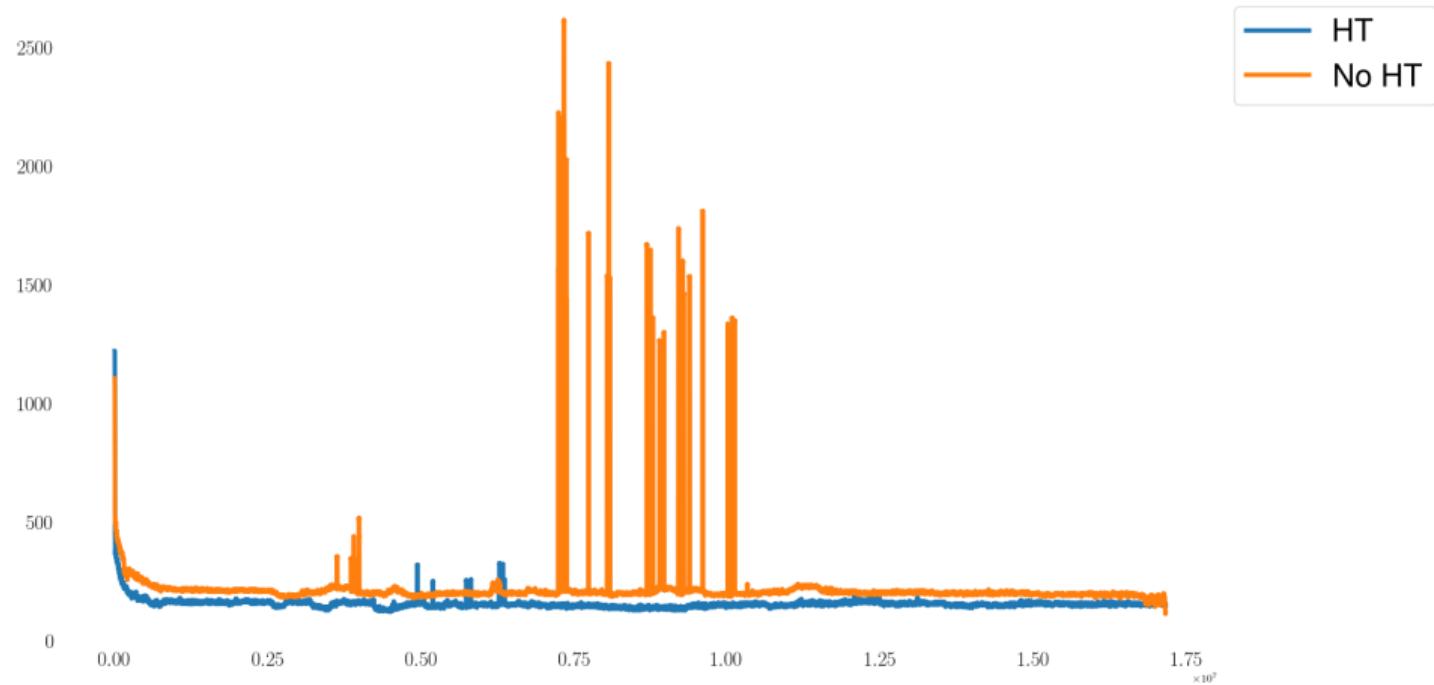
- Less deviation for latency
- Share execution state and cache
- Spin locks have significant impact
- PAUSE instruction (skylake latency 140 cycles)

Intel® 64 and IA-32 Architectures Optimization Reference Manual

Latency rolling standard deviation, r/w



Latency rolling standard deviation, readonly



Virtualization?

Timekeeping

Timekeeping in VMware Virtual: Information Guide

Timekeeping

- Statistical sampling
(occasional incorrect charging)

Timekeeping in VMware Virtual: Information Guide

Timekeeping

- Statistical sampling
(occasional incorrect charging)
- Exact measurement (TSC time drift)

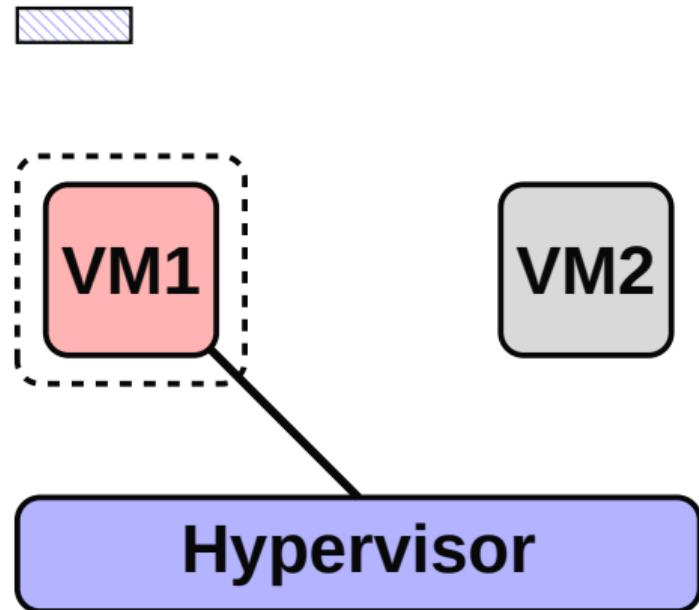
Timekeeping in VMware Virtual: Information Guide

Timekeeping

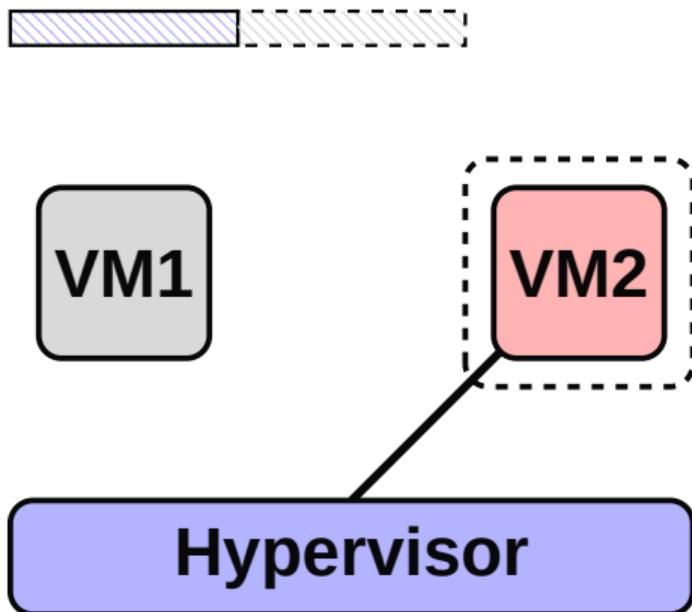
- Statistical sampling
(occasional incorrect charging)
- Exact measurement (TSC time drift)
- `/sys/devices/system/clocksource/clocksource0/`

Timekeeping in VMware Virtual: Information Guide

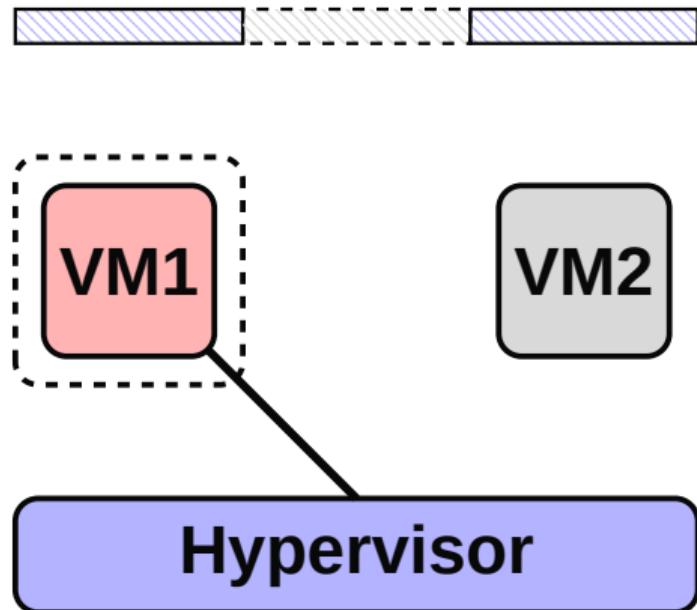
Scheduling



Scheduling



Scheduling

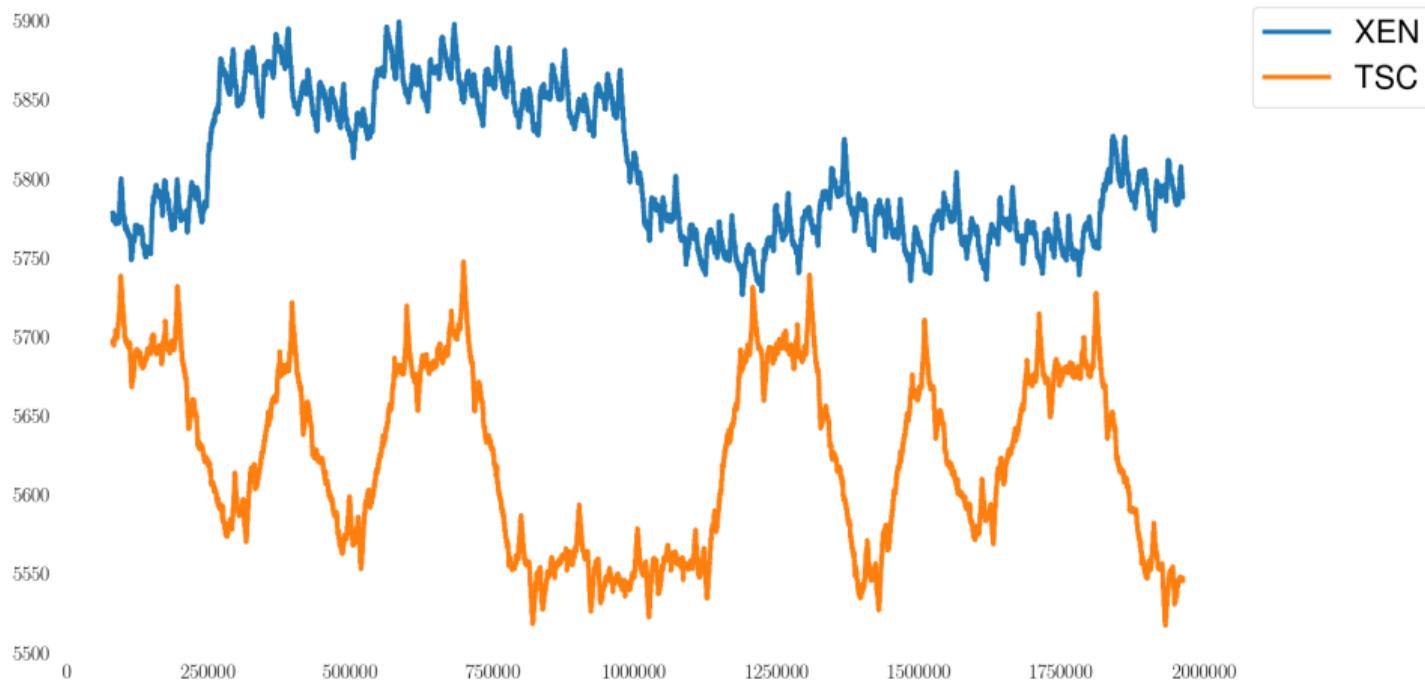


vDSO

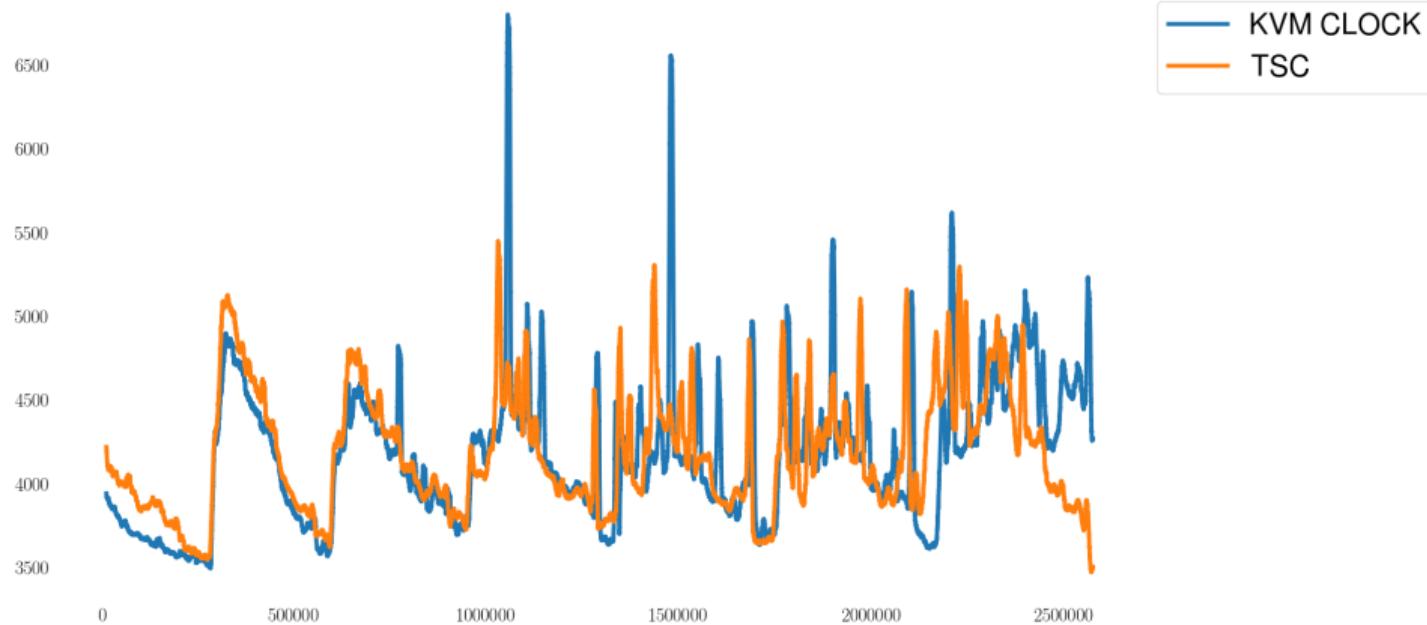
- gettimeofday
- clock_gettime
- XEN doesn't support vDSO for them
- unnecessary context switches to a kernel

Two frequently used system calls are 77% slower on AWS EC2

Latency m4.xlarge XEN/TSC, r/w

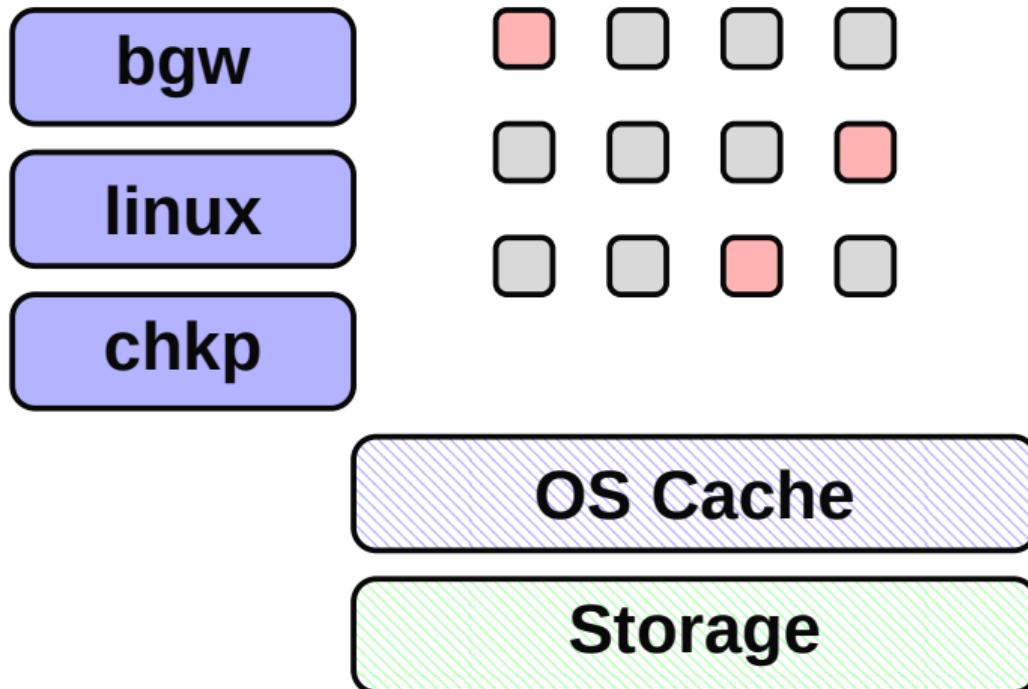


Latency m5.xlarge KVM/TSC, r/w

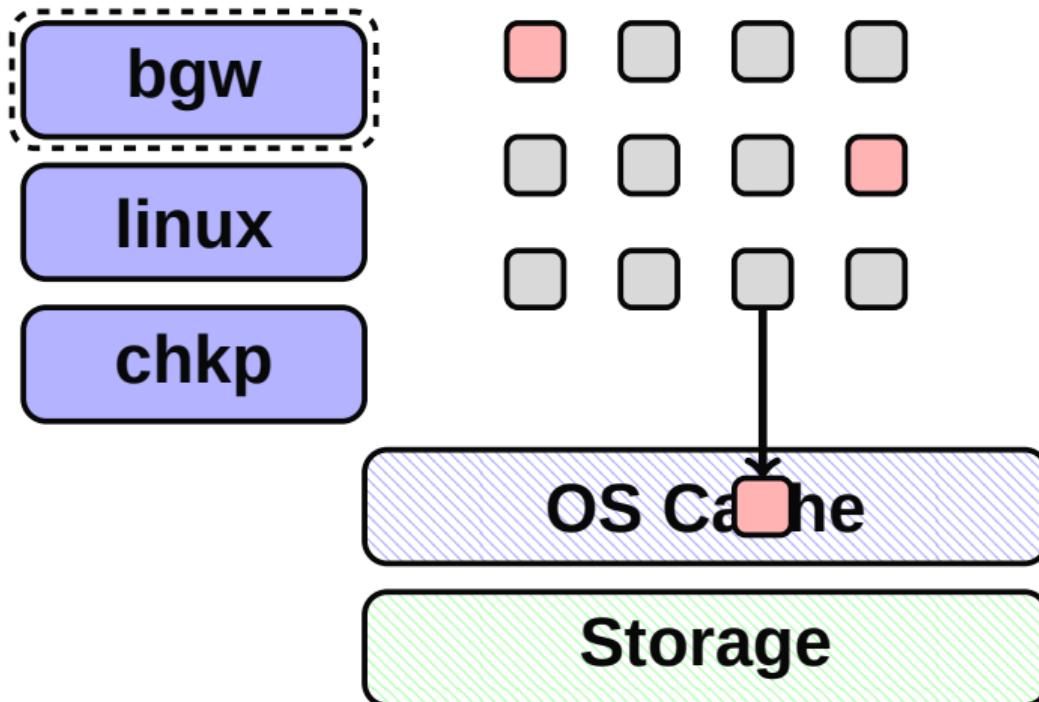


Memory

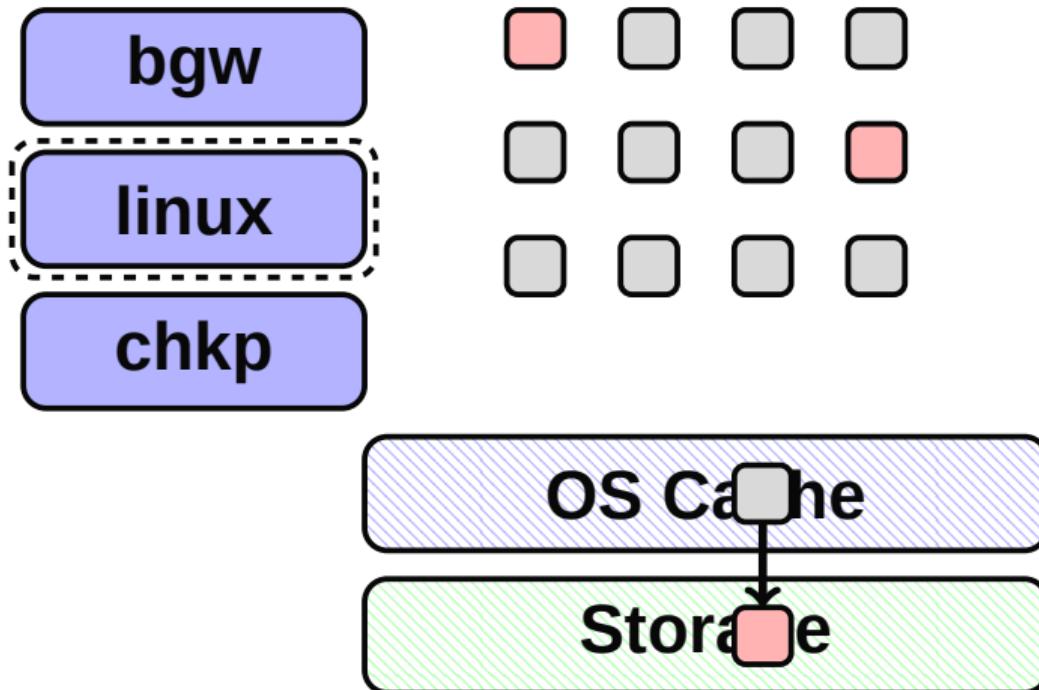
Dirty pages



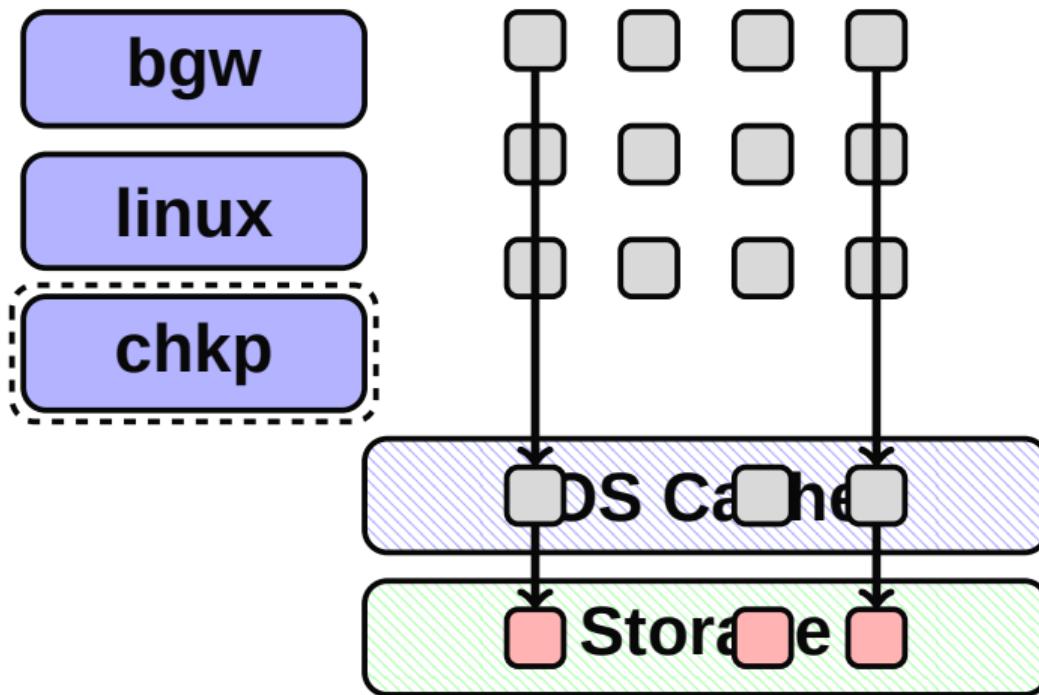
Dirty pages



Dirty pages



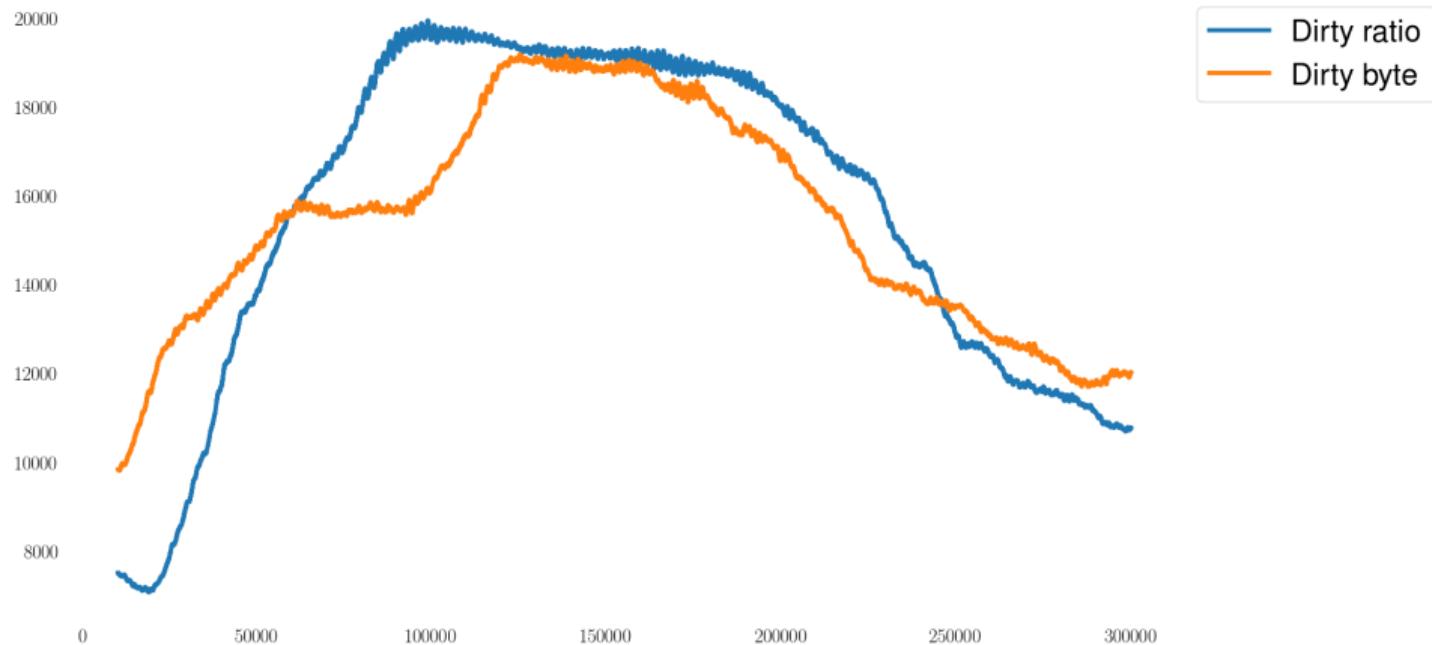
Dirty pages



Dirty pages, r/w

- vm.dirty_ratio 20
- vm.dirty_background_ratio 10
- vm.dirty_bytes 0
- vm.dirty_background_bytes 0

Dirty pages



Containers?

cgroups controllers

- cpu,cpuacct
- cpuset
- memory
- devices
- freezer
- net_cls
- rdma
- blkio
- perf_event
- net_prio
- hugetlb
- pids
- rdma

CONFIG_MEMCG_KMEM

- Enabled in modern versions
- PostgreSQL requires contiguous memory for shared buffers
- It's being allocated using slab
- `memory.kmem.limit_in_bytes` is too high

Host, normal

Zone: Normal

Free KiB in zone: 807232.00

Fragment size	Free fragments
4096	29612
8192	23308
16384	13495

Host with a container

Zone: Normal

Free KiB in zone: 109700.00

Fragment size	Free fragments
4096	3405
8192	7082
16384	1954

Bad neighbor

- buddy allocator can fail
to find a page of proper size
- kernel will start a compaction process
- compaction implementation
knows nothing about cgroups

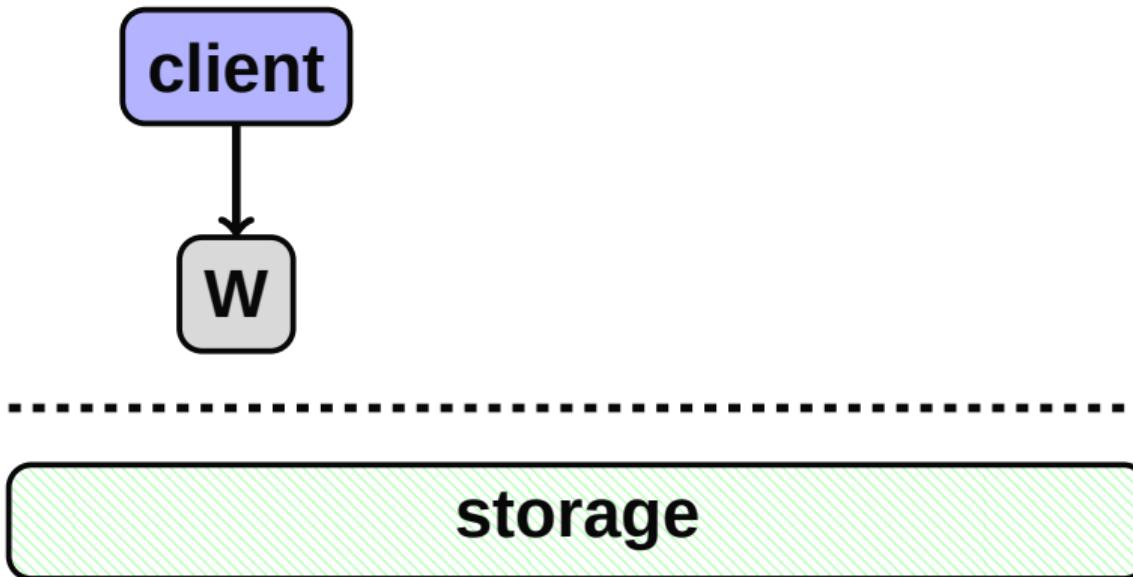
Storage IO

WAL

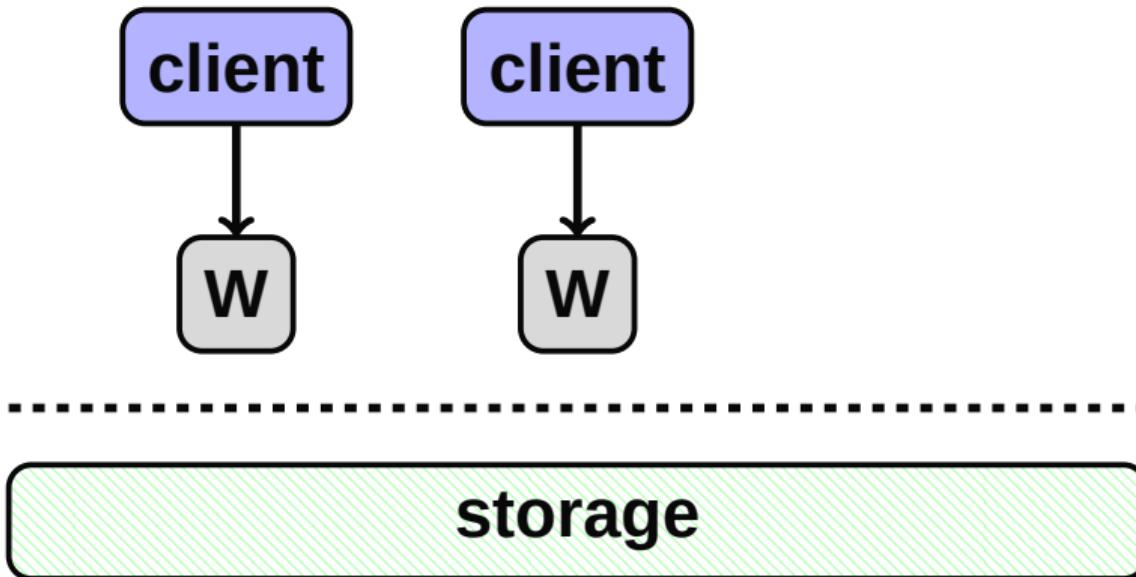
client

storage

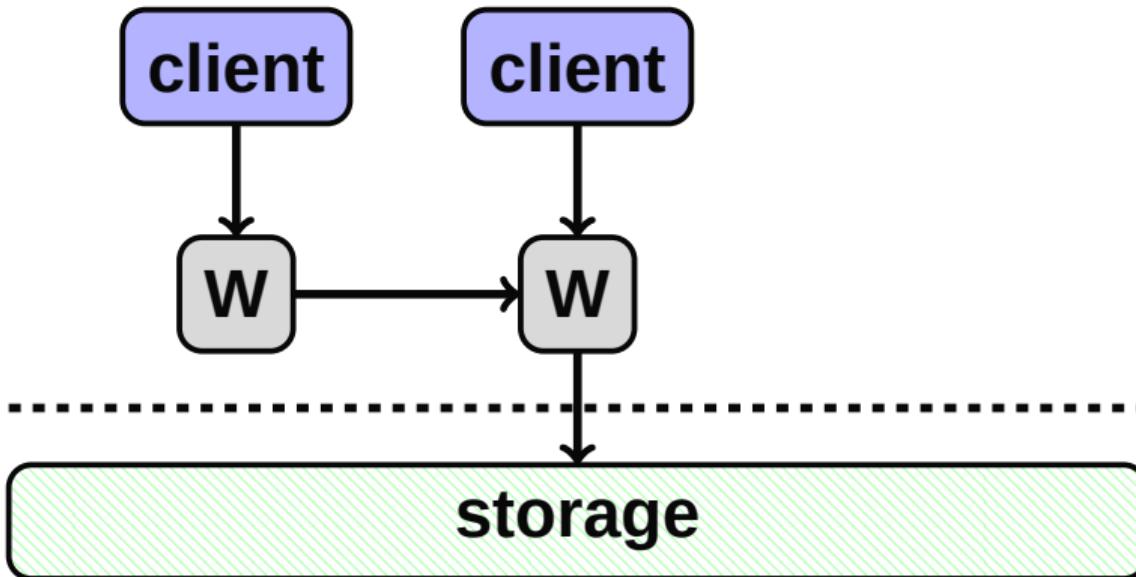
WAL



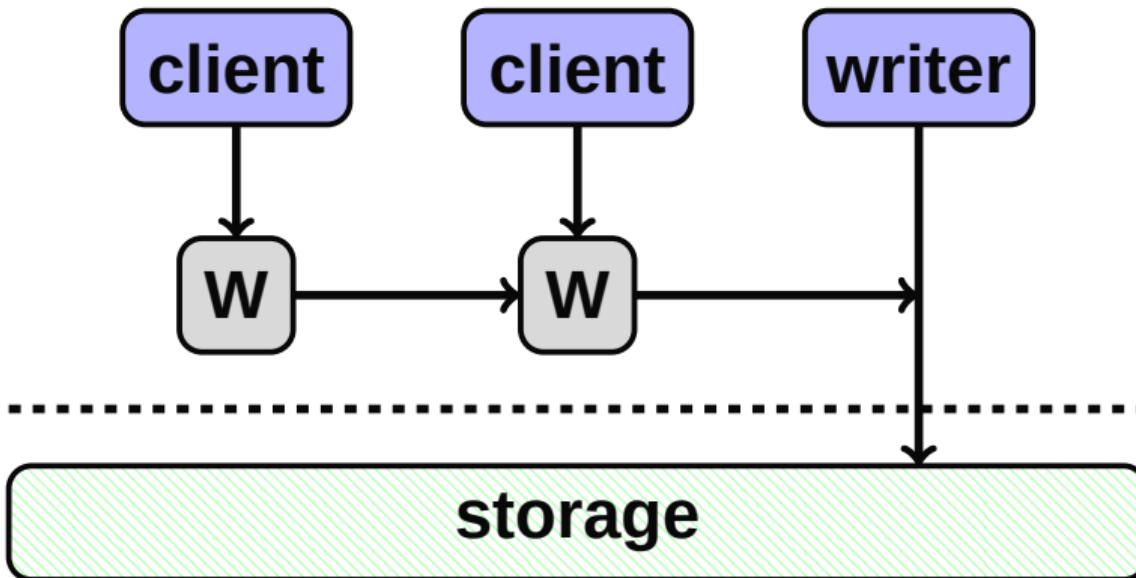
WAL



WAL



WAL



WAL

- bgwriter write to WAL
(transaction snapshots for replication)
- with logical decoding turned on
more data is going to WAL

NVMe

- better for resource sharing (PCI express) under the virtualization
- /sys/block/sda/queue/scheduler [noop|none]
- DSM operations

NVMe DSM

- Expected lifetime
- Sequential write/read prepare
- Access frequency

DSM support

- Command DWORD 11 in ioctl
- fcntl SET_FILE_RW_HINT
- nvme-cli
- Specify a start block and a range length

NVM Express Revision 1.3c May 24, 2018

```
# get a start block
hdparm -fibmap data_file
data_file:
  filesystem blocksize 4096, begins at LBA 0;
  assuming 512 byte sectors.
  byte_offset begin_LBA    end_LBA    sectors
          0      55041560    55041567          8

# set dsm for sequential read optimized
nvme dsm /dev/nvme1n01 -idr -slbs=55041560 -blocks=8
```

Containers?

blkio controller

- CFQ & throttling policy (generic block layer)
- No weight related options will work without CFQ
- Advisable io scheduler for SSD is noop/none
- Block layer do sampling to enforce throttling

```
8388  8388  postgres    blk_throtl_bio
      blk_throtl_bio+0x1          [kernel]
      dm_make_request+0x80        [kernel]
      generic_make_request+0xf6  [kernel]
      submit_bio+0x7d            [kernel]
      blkdev_issue_flush+0x68   [kernel]
      ext4_sync_file+0x310       [kernel]
      vfs_fsync_range+0x4b       [kernel]
      do_fsync+0x3d              [kernel]
      sys_fdatasync+0x13         [kernel]
      fdatasync+0x10              [libc-2.24.so]
      XLogBackgroundFlush+0x17e  [postgres]
      WalWriterMain+0x1cb        [postgres]
      PostmasterMain+0xfea       [postgres]
```

throttle_sample_time

This is the time window that blk-throttle samples data, in millisecond. blk-throttle makes decision based on the samplings. Lower time means cgroups have more smooth throughput, but higher CPU overhead. This exists only when CONFIG_BLK_DEV_THROTTLING_LOW is enabled.

blkio

On traditional cgroup hierarchies, relationships between different controllers cannot be established making it impossible for writeback to operate accounting for cgroup resource restrictions and all writeback IOs are attributed to the root cgroup.

<https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git>

Questions?

⌚ github.com/erthalion

⌚ github.com/erthalion/ansible-ycsb

🐦 [@erthalion](https://twitter.com/erthalion)

✉️ [dmitrii.dolgov at zalando dot de](mailto:dmitrii.dolgov@zalando.de)

✉️ [9erthalion6 at gmail dot com](mailto:9erthalion6@gmail.com)