PostgreSQL and Hugepages:

Working with an abundance of memory in modern servers

Fernando Laudares Camargos
fernando.laudares@percona.com
Engenheiro de suporte

José Zechel
zechel@gmail.com
DBA PostgreSQL

PGConf.Brasil 2019

PERCONA

Content

1. Motivation
2. How memory works
3. Working with larger pages
4. Large pages in practice
5. Testing
6. What I have learnt

Motivation

Understanding huge pages and how they affect databases

TokuDB, MongoDB and THP

```
2014-07-17 19:02:55 13865 [ERROR] TokuDB will not run with  transparent huge pages enabled.
2014-07-17 19:02:55 13865 [ERROR] Please disable them to continue.
2014-07-17 19:02:55 13865 [ERROR] (echo never > /sys/kernel/mm/transparent_hugepage/enabled)
```

Disable Transparent Huge Pages (THP)

Transparent Huge Pages (THP) is a Linux memory management system that reduces the overhead of Translation Lookaside Buffer (TLB) lookups on machines with large amounts of memory by using larger memory pages.

However, database workloads often perform poorly with THP, because they tend to have sparse rather than contiguous memory access patterns. You should disable THP on Linux machines to ensure best performance with MongoDB.

Source: https://docs.mongodb.com/manual/tutorial/transparent-huge-pages/

TokuDB, MongoDB and THP

```
2014-07-17 19:02:55 13865 [ERROR] TokuDB will not run with  transparent huge pages enabled.
2014-07-17 19:02:55 13865 [ERROR] Please disable them to continue.
2014-07-17 19:02:55 13865 [ERROR] (echo never > /sys/kernel/mm/transparent_hugepage/enabled)
```

Disable Transparent Huge Pages (THP)

Transparent Huge Pages (THP) is a Linux memory management system that reduces the overhead of Translation Lookaside Buffer (TLB) lookups on machines with large amounts of memory by using larger memory pages.

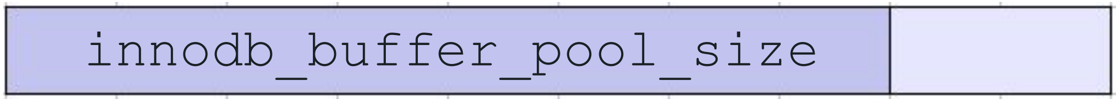
However, database workloads often perform poorly with THP, because they tend to have sparse rather than contiguous memory access patterns. You should disable THP on Linux machines to ensure best performance with MongoDB.

Source: https://docs.mongodb.com/manual/tutorial/transparent-huge-pages/

MySQL & PostgreSQL - database *cache*

- MySQL: InnoDB's Buffer Pool

The buffer pool is an area in main memory where caches table and index data as it is accessed. The buffer pool permits frequently used data to be processed directly from memory, which speeds up processing. On dedicated servers, up to 80% of physical memory is often assigned to the buffer pool. -- Source: https://dev.mysql.com/doc/refman/5.7/en/innodb-buffer-pool.html



PERCONA

MySQL & PostgreSQL - database *cache*

- PostgreSQL: shared memory buffers

If you have a dedicated database server with 1GB or more of RAM, a reasonable starting value for shared_buffers is 25% of the memory in your system. There are some workloads where even larger settings for shared_buffers are effective, but because PostgreSQL also relies on the operating system cache, it is unlikely that an allocation of more than 40% of RAM to shared_buffers will work better than a smaller amount.

-- Source: https://www.postgresql.org/docs/10/runtime-config-resource.html

MySQL & PostgreSQL - database *cache*

- PostgreSQL: shared memory buffers

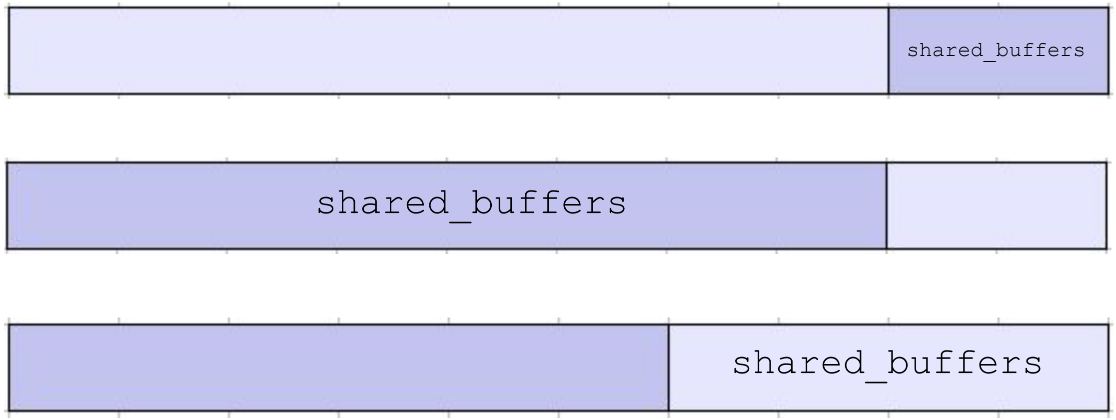
If you have a dedicated database server with 1GB or more of RAM, a reasonable starting value for shared_buffers is 25% of the memory in your system. There are some workloads where even larger settings for shared_buffers are effective, but because PostgreSQL also relies on the operating system cache, it is unlikely that an allocation of more than 40% of RAM to shared_buffers will work better than a smaller amount.

-- Source: https://www.postgresql.org/docs/10/runtime-config-resource.html

MySQL & PostgreSQL - database *cache*

- PostgreSQL: shared memory buffers

  Does the *dataset* fit in memory?



PERCONA

How memory works

A very brief overview of memory management

In a nutshell

1. Applications (and the OS) run in ***virtual memory***

Every process is given the impression that it is working with large, contiguous sections of memory

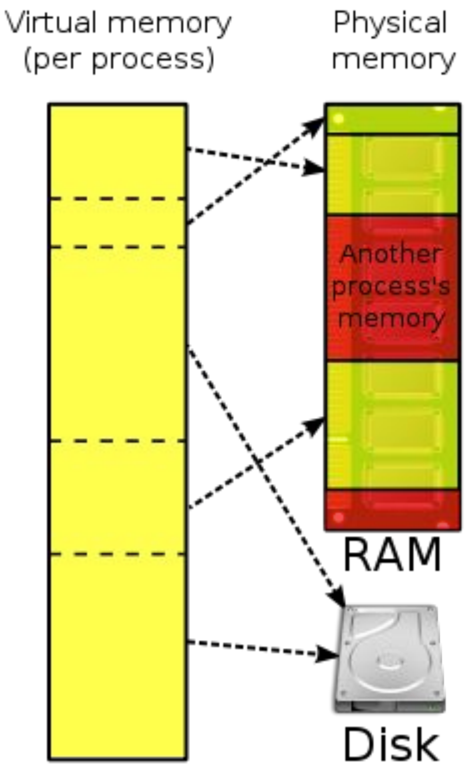


Image source: https://en.wikipedia.org/wiki/Virtual_memory

In a nutshell

2. Virtual memory is ***mapped*** into physical memory by the OS using a ***page table***

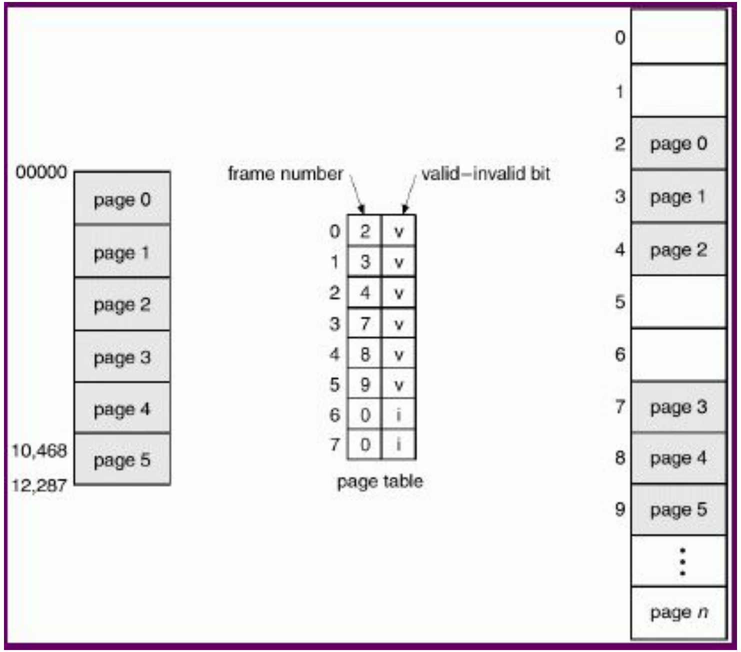


Image source: http://courses.teresco.org/cs432_f02/lectures/12-memory/12-memory.html

In a nutshell

3. The address translation logic is implemented by the **MMU**

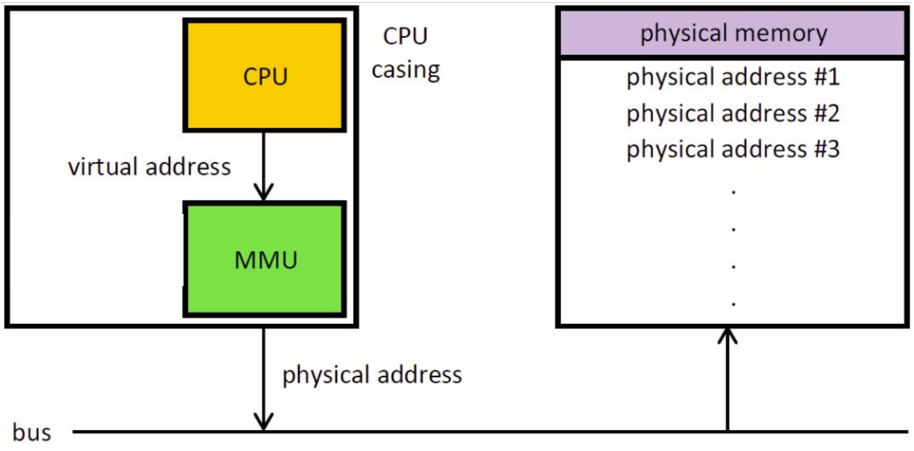
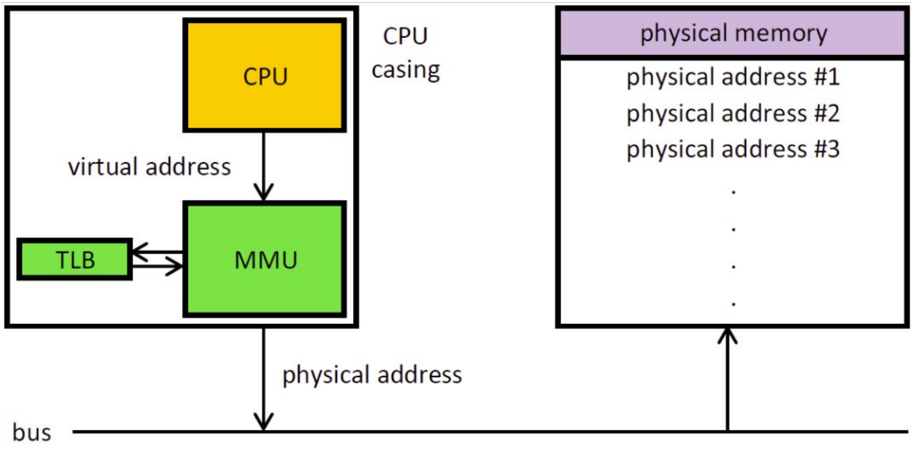


Image adapted from https://en.wikipedia.org/wiki/Memory_management_unit

In a nutshell

4. The MMU employs a cache of recently used pages known as ***TLB***



Translation Lookaside Buffer

Image adapted from https://en.wikipedia.org/wiki/Memory_management_unit

In a nutshell

5. The TLB is searched first:

- if a match is found the physical address of the page is returned → **TLB hit**

1 memory access

- else scan the page table (*walk*) looking for the address mapping (entry) → **TLB miss**

"2" memory accesses

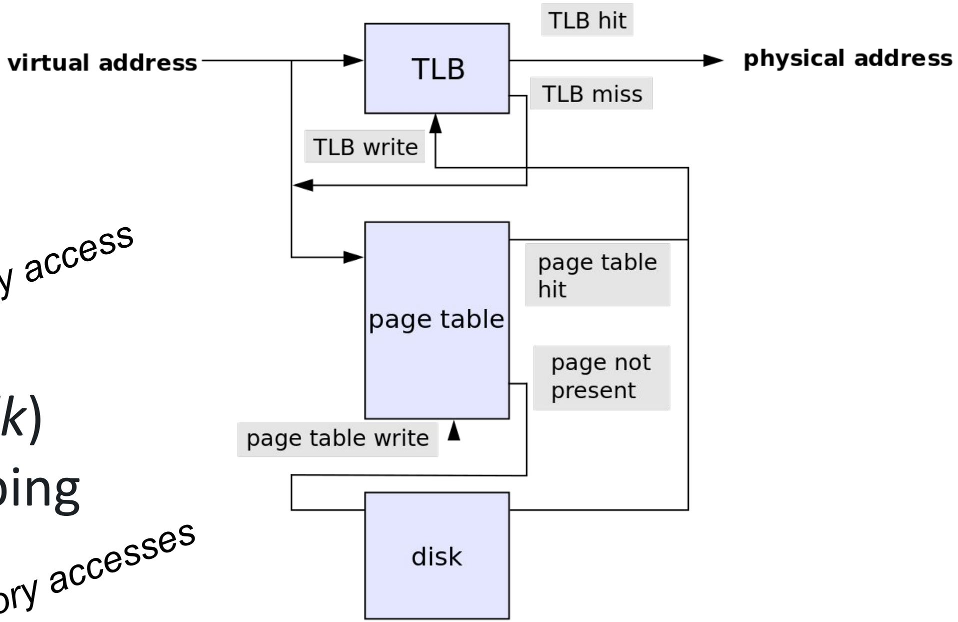


Image source: https://en.wikipedia.org/wiki/Page_table

Constraint

TLB can only cache a few hundred entries

How can we improve its efficiency (decrease *misses*?)

A. Increase TLB size → expensive
B. Increase page size → less pages to map

Inspiration: https://alexandrnikitin.github.io/blog/transparent-hugepages-measuring-the-performance-impact/

Page sizes & TLB

- Typical page size is 4K
- Many modern processors support other page sizes

If we consider a server with 256G of RAM:

4K	67108864
2M	131072
1G	256

large/huge pages

PERCONA

Working with larger pages

Employing huge pages in PostgreSQL

Why?

The main premise is:

Less page table lookups, more "performance"

How?

Two ways:

1. Application has native support for working with huge pages
   Ex: JVM, MySQL, PostgreSQL

PostgreSQL

"Using huge pages reduces overhead when using large contiguous chunks of memory, as PostgreSQL does, particularly when using large values of `shared_buffers`*."*

Source: https://www.postgresql.org/docs/9.4/kernel-resources.html#LINUX-HUGE-PAGES

How?

The other way is:

2. "Blindly"

- Application does not have support for huge pages…
  … but the underlying OS (Linux) does:

Transparent Huge Pages

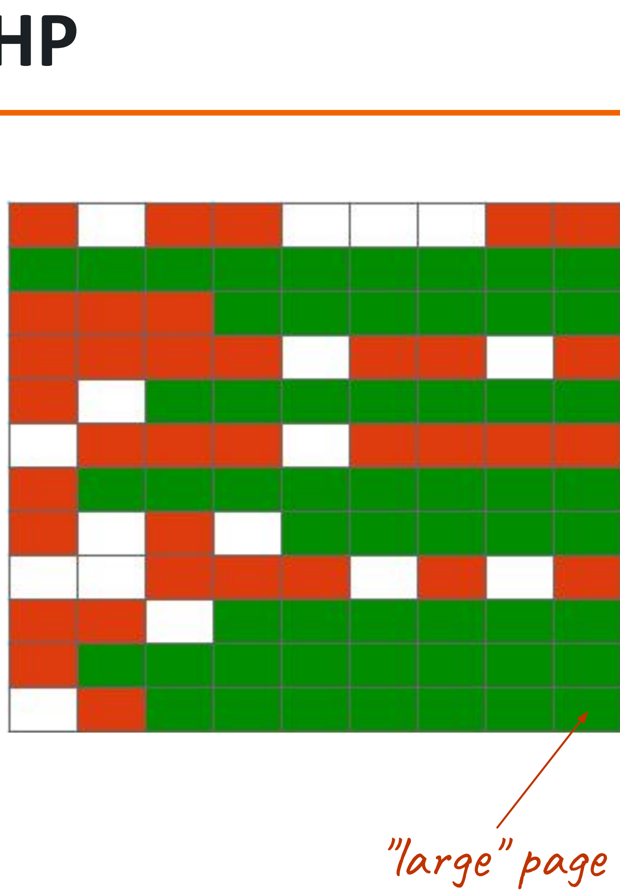
THP

The kernel works in the background (`khugepaged`) trying to:

- "create" huge pages
  - find enough contiguous blocks of memory
  - "convert" them into a huge page
- *transparently* allocate them to processes when there is a "fit"
  - shouldn't provide a 2M-page for someone asking 128K

PERCONA

THP

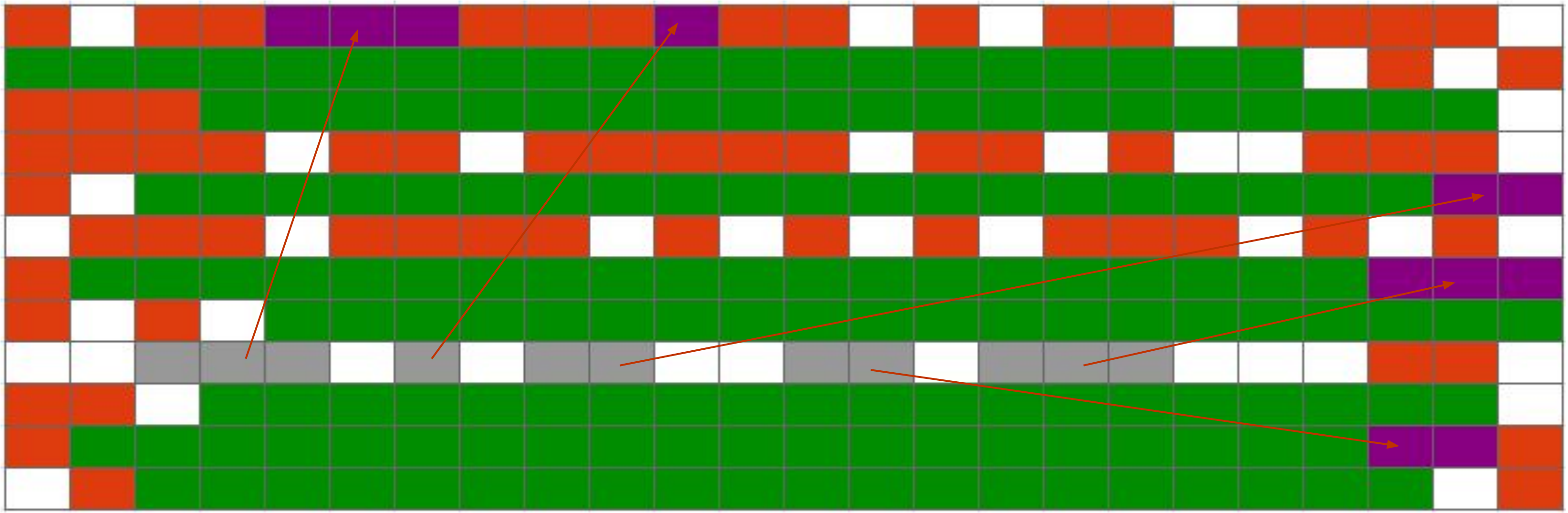


PERCONA

THP

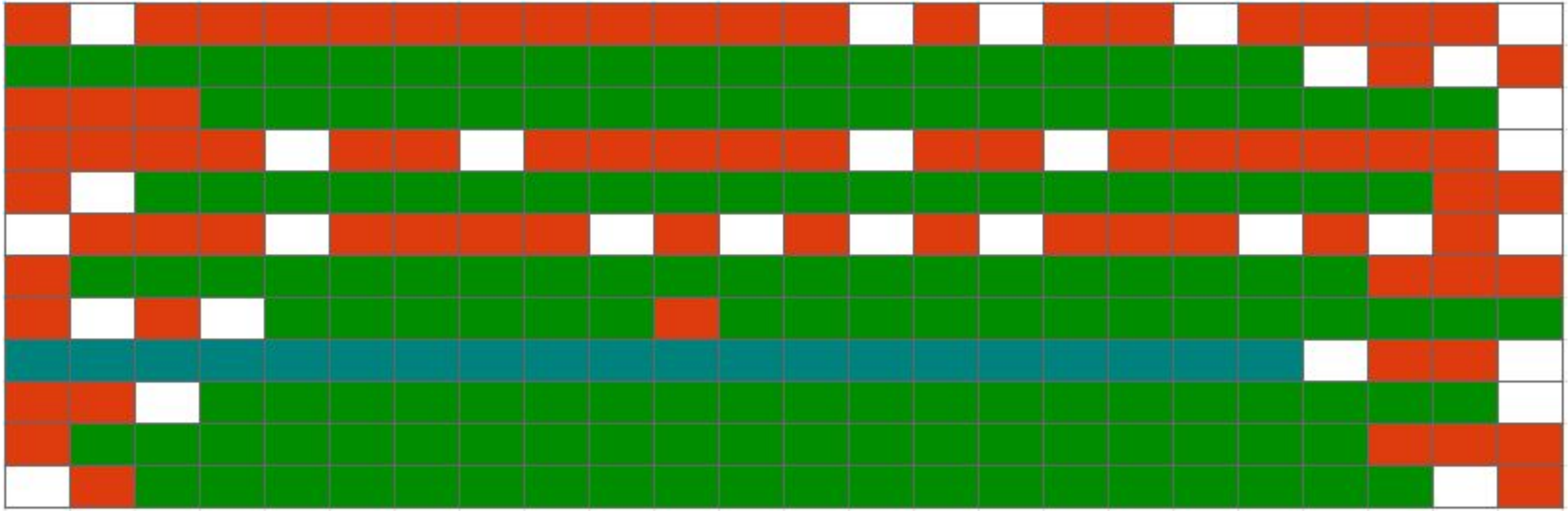
PERCONA

THP



PERCONA

THP



THP

THP

`khugepaged` work is somewhat expensive and <u>may</u> cause stalls

- known to cause latency spikes in certain situations
  - pages are locked during their manipulation

Huge pages in practice

How to do it

Architecture support for huge pages

```
# cat /proc/cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family      : 6
model           : 63
model name      : Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz
(...)
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf
eagerfpu pni pclmulqdq dtes64 ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid
dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand
lahf_lm abm epb tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2
smep bmi2 erms invpcid cqm xsaveopt cqm_llc cqm_occup_llc dtherm ida arat pln pts
```

Architecture support for huge pages

```
# cat /proc/cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family      : 6
model           : 63
model name      : Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz
(...)
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr p 1G mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm
constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf
eagerfpu pni pclmulqdq dtes64 ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid
dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand
lahf_lm abm epb tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2
smep bmi2 erms invpcid cqm xsaveopt cqm_llc cqm_occup_llc dtherm ida arat pln pts
```

2M (pse)

1G (pdpe1gb)

PERCONA

Architecture support for huge pages

```
# cat /proc/meminfo
MemTotal:          264041660 kB
(...)
Hugepagesize:           2048 kB
DirectMap4k:          128116 kB
DirectMap2M:         3956736 kB
DirectMap1G:       266338304 kB
```


PERCONA

Changing huge page size

1) `# vi /etc/default/grub`

```
GRUB_CMDLINE_LINUX_DEFAULT="hugepagesz=1GB default_hugepagesz=1G"
```

2) `# update-grub`

```
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-4.4.0-75-generic
Found initrd image: /boot/initrd.img-4.4.0-75-generic
Found memtest86+ image: /memtest86+.elf
Found memtest86+ image: /memtest86+.bin
done
```

3) `# shutdown -r now`

PERCONA

Creating a "pool" of huge pages

```
# sysctl -w vm.nr_hugepages=10

# cat /proc/meminfo | grep Huge
AnonHugePages:      2048 kB
HugePages_Total:      10
HugePages_Free:       10
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:    1048576 kB

# free -m
             total        used        free      shared  buff/cache   available
Mem:        257853         776      256938           9         137      256319
...
Mem:        257853       11007      246705           9         140      246087
```

11007M - 776M = 9.99G

PERCONA

Creating a "pool" of huge pages - NUMA

```
# numastat -cm | egrep 'Node|Huge'
                 Node 0 Node 1  Total
AnonHugePages         2      0      2
HugePages_Total    5120   5120  10240
HugePages_Free     5120   5120  10240
HugePages_Surp        0      0      0
```


PERCONA

Creating a "pool" of huge pages - in a single node

```
# sysctl -w vm.nr_hugepages=0

# echo 10 > /sys/devices/system/node/node0/hugepages/hugepages-1048576kB/nr_hugepages

# numastat -cm | egrep 'Node|Huge'
                 Node 0 Node 1  Total
AnonHugePages         2      0      2
HugePages_Total   10240      0  10240
HugePages_Free    10240      0  10240
HugePages_Surp        0      0      0
```

"Online" huge page allocation

It might not work!

```
# sysctl -w vm.nr_hugepages=256
vm.nr_hugepages = 256

# cat /proc/meminfo | grep Huge
AnonHugePages:       2048 kB
HugePages_Total:      246
HugePages_Free:       246
HugePages_Rsvd:         0
HugePages_Surp:         0
Hugepagesize:     1048576 kB
```

Allocating huge pages at boot time

```
GRUB_CMDLINE_LINUX_DEFAULT="hugepagesz=1GB default_hugepagesz=1G
                                   hugepages=100"
```


PERCONA

Disabling THP

```
# cat /proc/meminfo | grep AnonHuge
AnonHugePages:      2048 kB

# ps aux |grep huge
root         42  0.0  0.0      0     0 ?        SN   Jan17   0:00 [khugepaged]
```

To disable it:

- at runtime:

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled
# echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

- at boot time:

```
GRUB_CMDLINE_LINUX_DEFAULT="(...) transparent_hugepage=never "
```

Configuring database

Userland

Give the user permission to use huge pages ...

1) `# getent group mysql`
   `mysql:x:`**`1001`**`:`

2) `# echo `**`1001`**` > /proc/sys/vm/hugetlb_shm_group`

Limits

… and/or give the user permission to *lock* (enough) memory:

1) `# cp /lib/systemd/system/mysql.service /etc/systemd/system/`

2) `# vim /etc/systemd/system/mysql.service`

```
[Service]
...
LimitMEMLOCK=infinity
```

3) `# systemctl daemon-reload`

PERCONA

Enabling huge pages in the database

MySQL

```
# vim /etc/mysql/my.cnf
```

```
[mysqld]
...
large_pages=ON
```

```
# service mysql restart
```

PostgreSQL

```
# vim /etc/postgresql/10/main/postgresql.conf
```

```
huge_pages=ON
```

```
# service postgresql restart
```

Testing

Experimenting popular database benchmarks with huge pages

At first

- Curious about how huge pages would affect "performance"
- Less interested in measuring TLB improvements

Plan

- Test with popular benchmarks with PostgreSQL
  - Sysbench-TPCC, Sysbench-OLTP, pgBench
- Consider two situations:
  - Dataset fits in memory (Buffer Pool / shared_buffers)
  - Dataset does **not** fit in memory
- Run each test three times:
  - With regular 4K pages as baseline, then 2M & 1G huge pages
- Run each test with different number of clients (threads):
  - 56, 112, 224, 448

Test server

Hardware

- Intel Xeon E5-2683 v3 @ 2.00GHz
  - 2 sockets = 28 cores, 56 threads
- 256GB of RAM
- Samsung SM863 SSD, 1.92TB (EXT4)

OS

- Ubuntu 16.04.2 LTS
  - Linux 4.4.0-75-generic #96-Ubuntu SMP

Databases

- PostgreSQL 10 (10.6-1.pgdg16.04+1)

Benchmarks

- Sysbench 1.1.0-7df3892, Sysbench-TPCC
- pgBench (Ubuntu 10.6-1.pgdg16.04+1)

Database configuration

PostgreSQL

```
max_connections = 1000
maintenance_work_mem = 1GB
bgwriter_lru_maxpages = 1000
bgwriter_lru_multiplier = 10.0
bgwriter_flush_after = 0
wal_level = minimal
fsync = on
synchronous_commit = on
wal_sync_method = fsync
full_page_writes = on
wal_compression = on
checkpoint_timeout = 1
checkpoint_completion_target = 0.9
max_wal_size = 200GB
min_wal_size = 1GB
max_wal_senders = 0
random_page_cost = 1.0
effective_cache_size = 100GB
log_checkpoints = on
autovacuum_vacuum_scale_factor = 0.4
shared_buffers = XXXGB
huge_pages = X
```

varying

PERCONA

Double check during initialization - PostgreSQL

```
huge_pages = on
```

```
2019-01-17 09:46:10.138 EST [20982]FATAL:  could not map anonymous shared memory: Cannot allocate memory
2019-01-17 09:46:10.138 EST [20982] HINT:  This error usually means that PostgreSQL's request for a shared
memory segment exceeded available memory, swap space, or huge pages. To reduce the request size (currently
184601698304 bytes), reduce PostgreSQL's shared memory usage, perhaps by reducing shared_buffers or
max_connections.
2019-01-17 09:46:10.138 EST [20982] LOG:  database system is shut down
```


PERCONA

Benchmarks

Sysbench-TPCC: PostgreSQL

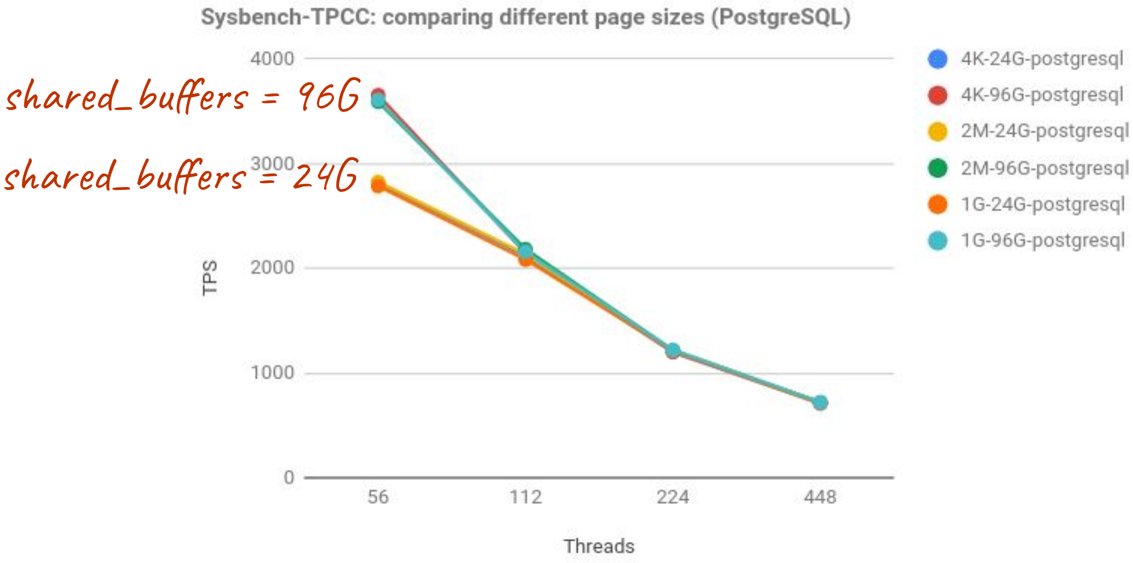
- Prepare:

```
Sysbench tpcc.lua --db-driver=pgsql --pgsql-db=sysbench --pgsql-user=sysbench --pgsql-password=sysbench
--threads=56 --report-interval=1--tables=10 --scale=100 --use_fk=0 --trx_level=RC prepare
```

- Run:

```
sysbench tpcc.lua --db-driver=pgsql --pgsql-host=localhost--pgsql-port=5432 --pgsql-db=sysbench
--pgsql-user=sysbench --pgsql-password=sysbench--threads=X --report-interval=1 --tables=10 --scale=100
--use_fk=0 --trx_level=RC --time=3600 run
```

Sysbench-TPCC: PostgreSQL



Sysbench OLTP point_select: PostgreSQL

- Prepare:

```
$ sysbench oltp_point_select.lua --db-driver=pgsql --pgsql-host=localhost --pgsql-db=sysbench
--pgsql-user=sysbench --pgsql-password=sysbench --threads=56 --report-interval=1--tables=10
--table-size=80000000 prepare
```

```
$ vacuumdb sysbench
```

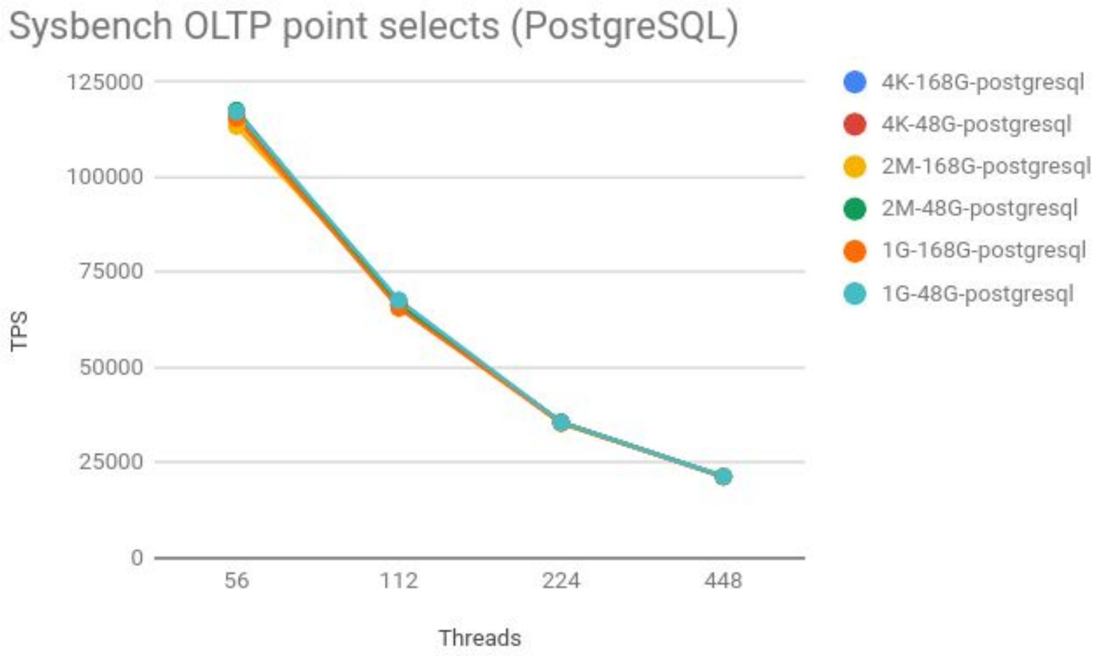
Resulting:

```
sysbench=# SELECT datname, pg_size_pretty(pg_database_size(datname)), blks_read,
blks_hit, temp_files, temp_bytes from pg_stat_database where datname='sysbench';
 datname  | pg_size_pretty | blks_read |  blks_hit  | temp_files | temp_bytes
----------+----------------+-----------+------------+------------+-------------
 sysbench | 198 GB         |  37777656 | 4478661433 |         20 | 16031580160
```

- Run:

```
$ sysbench oltp_point_select.lua --db-driver=pgsql --pgsql-host=localhost --pgsql-port=5432
--pgsql-db=sysbench --pgsql-user=sysbench --pgsql-password=sysbench --threadX=--report-interval=1
--tables=10 --table-size=80000000 --time=3600 run
```

Sysbench OLTP point selects: PostgreSQL



PERCONA

pgBench select-only: PostgreSQL

- Prepare:

```
$ pgbench --username=sysbench --host=localhost -i--scale=12800 sysbench
```

Resulting:

```
sysbench=# SELECT datname, pg_size_pretty(pg_database_size(datname)), blks_read,
blks_hit, temp_files, temp_bytes from pg_stat_database where datname='sysbench';
 datname  | pg_size_pretty | blks_read | blks_hit | temp_files | temp_bytes
----------+----------------+-----------+----------+------------+-------------
 sysbench | 187 GB         |  62983477 | 21142806 |         24 | 25650487296
(1 row)
```

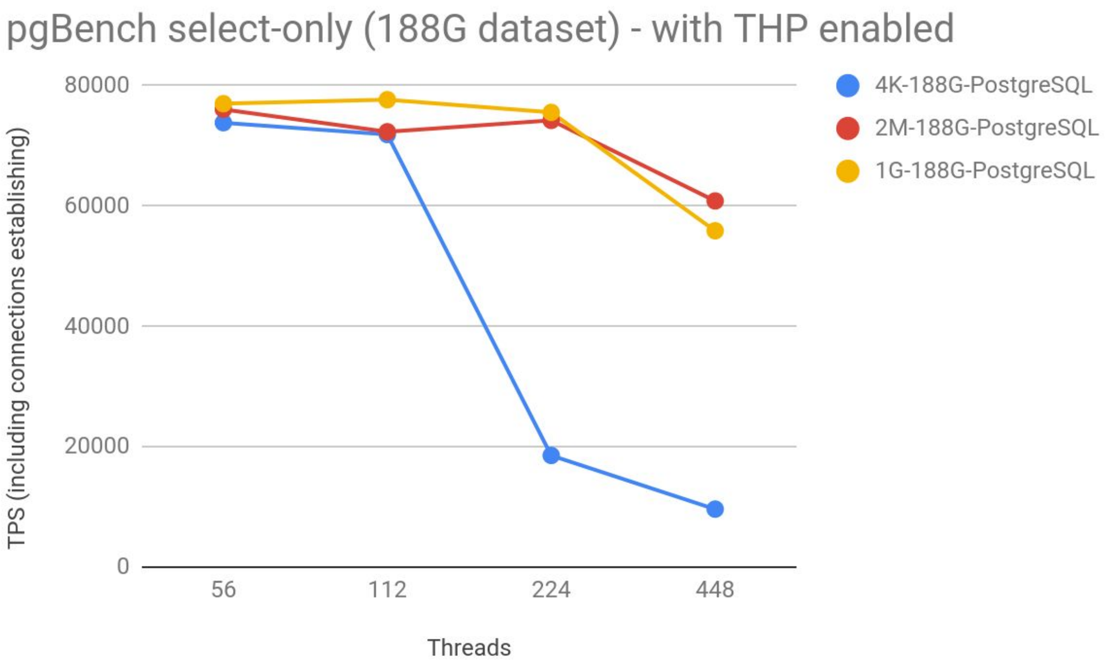
- Run:

```
$ pgbench --username=sysbench --host=localhost --builtin=select-only --clienX=--no-vacuum --time=3600
--progress=1 sysbench
```


PERCONA

pgBench select-only: PostgreSQL

pgBench select-only: PostgreSQL with THP enabled



What about *efficiency* ?

From Mark Callaghan's:

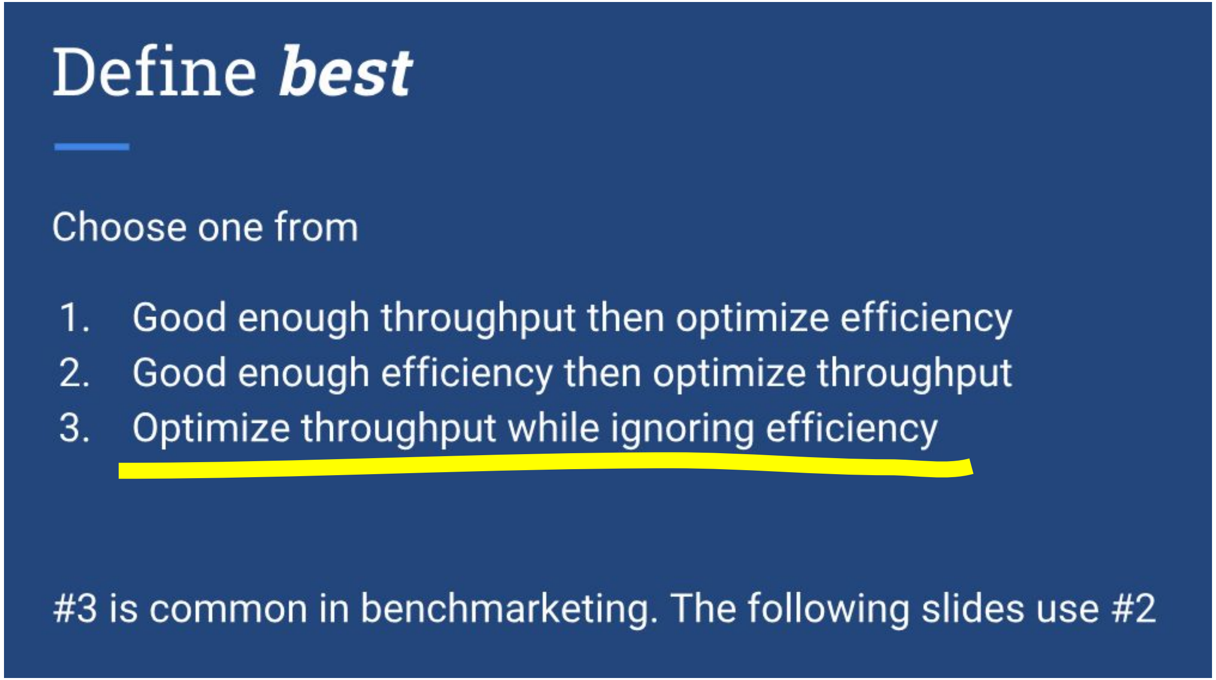
[*Efficiency vs performance - Use the right index structure for the job*]

In his quest for finding:

- the best configuration of the best index structure (for LSM)

Considering:

- performance goals
- constraints on hardware and efficiency



Source: http://smalldatum.blogspot.com/2019/01/optimal-configurations-for-lsm-and-more.html

PERCONA

Measuring efficiency directly

- Using large pages to improve the effectiveness of the TLB
    - by increasing the page size there will be less pages to map
    - <u>should be visible at the CPU level</u>
        - *CPU shall have less work to do*

PERCONA

Measuring CPU counters with Perf

1) Perf has built-in event aliases for counters of type `.MISS_CAUSES_A_WALK` at the TLB level:

- Data
  - dTLB-loads
  - dTLB-load-misses
  - dTLB-stores
  - dTLB-store-misses
- Instructions
  - iTLB-load
  - iTLB-load-misses

Inspiration: https://alexandrnikitin.github.io/blog/transparent-hugepages-measuring-the-performance-impact/

Measuring CPU counters with Perf

2) Number of CPU cycles spent in the page table walking:

- `cycles`
- `cpu/event=0x08,umask=0x10,name=dcycles`
- `cpu/event=0x85,umask=0x10,name=icycles`

Measuring CPU counters with Perf

3) Number of main memory reads caused by TLB miss:

- `cache-misses`
- `cpu/event=0xbc,umask=0x18,name=dreads`
- `cpu/event=0xbc,umask=0x28,name=ireads`

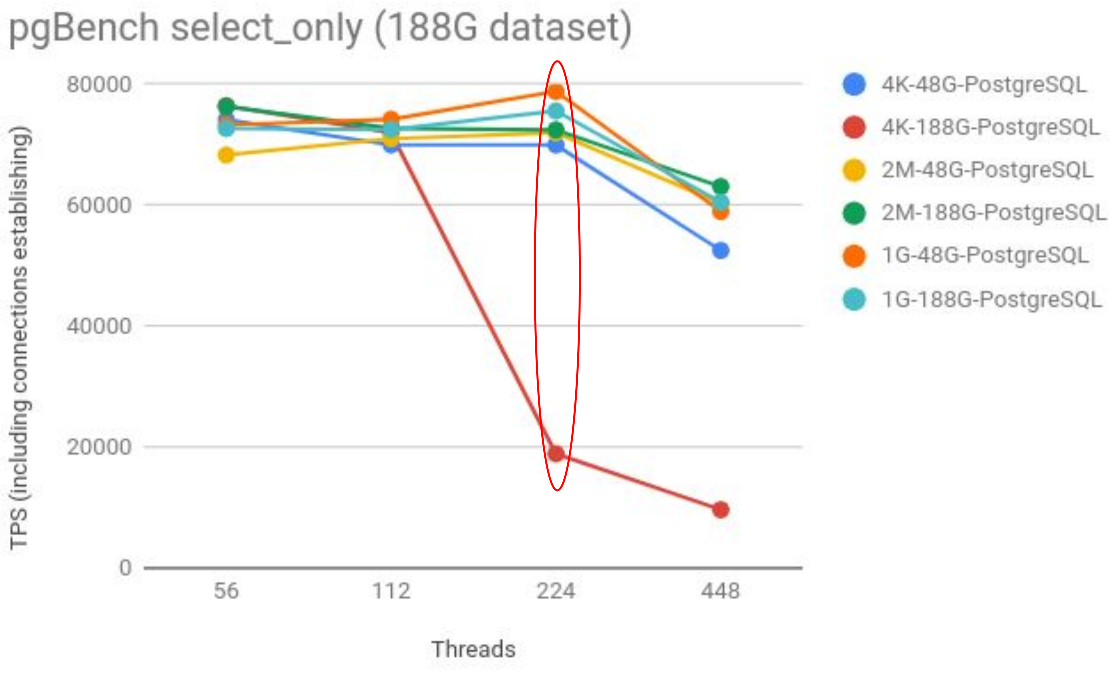
Measuring CPU counters with Perf

```
sudo perf stat -e dTLB-loads,dTLB-load-misses,dTLB-stores,dTLB-store-misses -e
iTLB-load,iTLB-load-misses -e cycles -e cpu/event=0x08,umask=0x10,name=dcycles/ -e
cpu/event=0x85,umask=0x10,name=icycles/ -e cpu/event=0xbc,umask=0x18,name=dreads/
-e cpu/event=0xbc,umask=0x18,name=dreads/ -e cpu/event=0xbc,umask=0x28,name=ireads/
-p 2525 sysbench oltp_point_select.lua --db-driver=mysql --mysql-host=localhost
--mysql-socket=/var/run/mysqld/mysqld.sock --mysql-db=sysbench
--mysql-user=sysbench --mysql-password=sysbench --threads=448 --report-interval=1
--tables=10 --table-size=80000000 --time=3600 run
```

mysqld

PERCONA

Measuring CPU counters with Perf

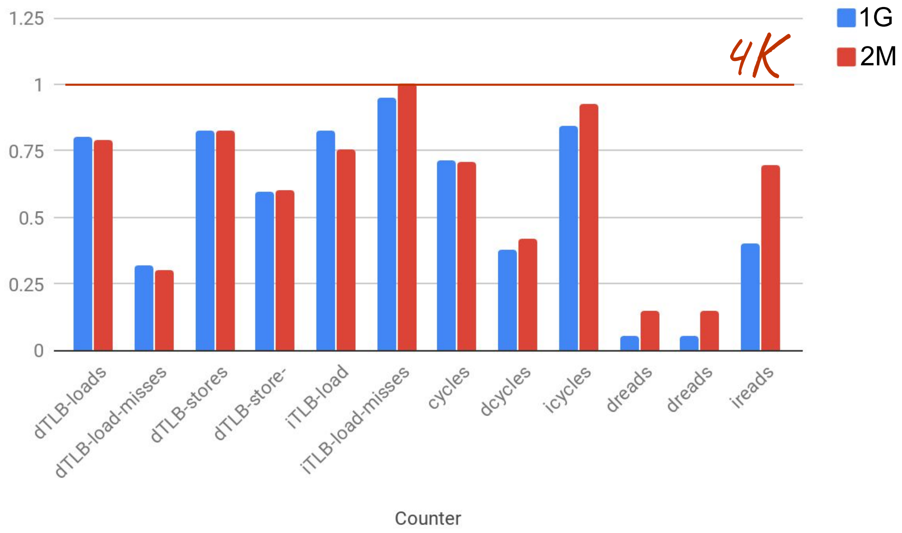


Counter	4K	1G
dTLB-loads	25.42%	27.91%
dTLB-load-misses	22.44%	25.90%
misses/hits	1.73%	0.69%
dTLB-stores	19.32%	19.99%
dTLB-store-misses	18.15%	18.14%
iTLB-load	18.45%	18.36%
iTLB-load-misses	24.74%	24.89%
misses/hits	152.29%	175.49%
cycles	32.74%	33.01%
dcycles	32.70%	32.95%
icycles	32.67%	32.95%
dreads	32.59%	32.90%
dreads	32.64%	32.94%
ireads	32.68%	32.97%

PERCONA

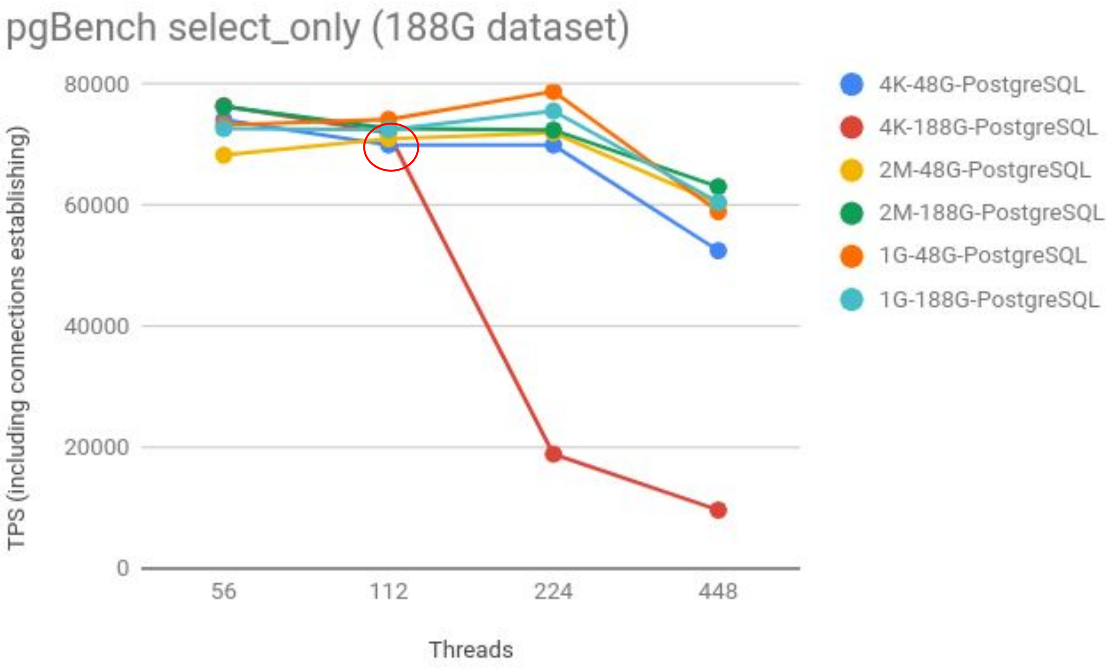
Measuring CPU counters with Perf

Counter	4K	1G
dTLB-loads	3,962,945,638,615	12,233,862,113,582
dTLB-load-misses	68,542,660,649	84,202,669,649
dTLB-stores	2,673,374,398,091	8,516,022,476,175
dTLB-store-misses	4,111,585,610	9,393,469,775
iTLB-load	21,975,305,991	69,718,900,178
iTLB-load-misses	33,465,650,082	122,349,897,580
cycles	26,842,071,449,916	73,689,973,037,599
dcycles	2,195,701,733,827	3,176,903,465,922
icycles	1,143,500,465,054	3,713,191,066,587
dreads	1,786,865,020	376,718,232
dreads	1,789,155,994	377,117,625
ireads	559,924,613	866,309,693
transactions	68077576	261651611



PERCONA

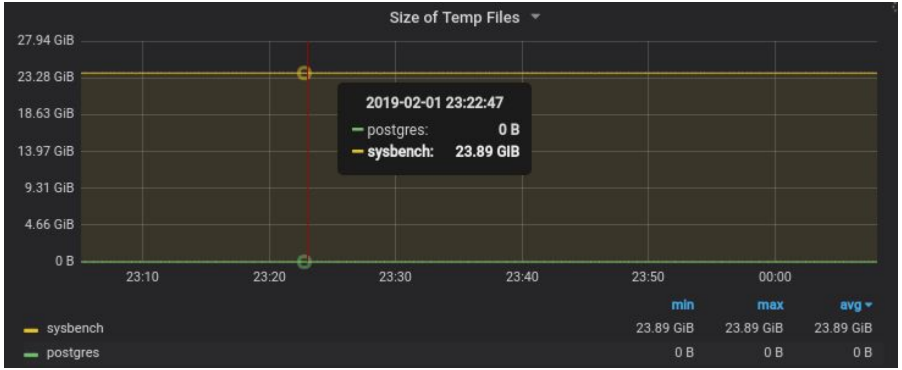
pgBench select-only: PostgreSQL

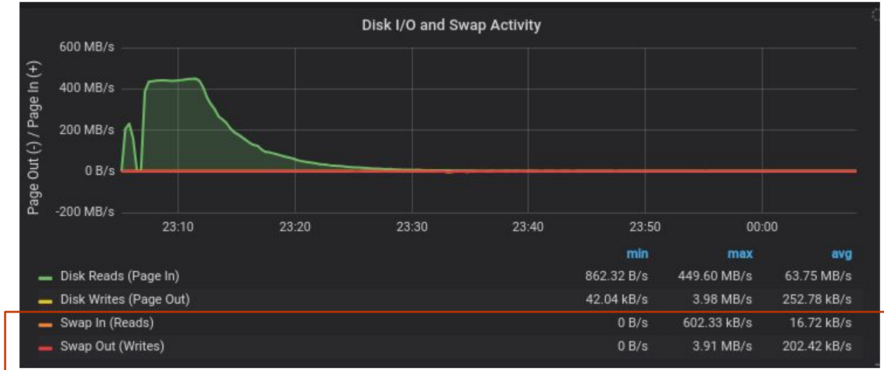


PERCONA

pgBench select-only: PostgreSQL

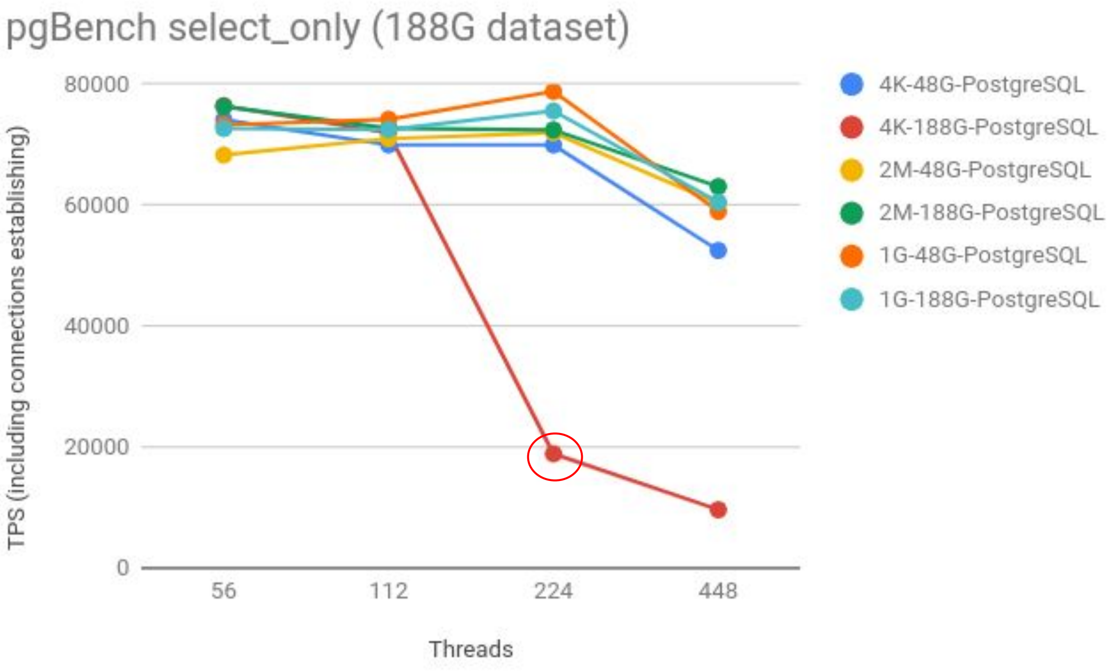
- `4K-pages, 188G shared_buffers, 112 clients`





PERCONA

pgBench select-only: PostgreSQL



pgBench select-only: PostgreSQL

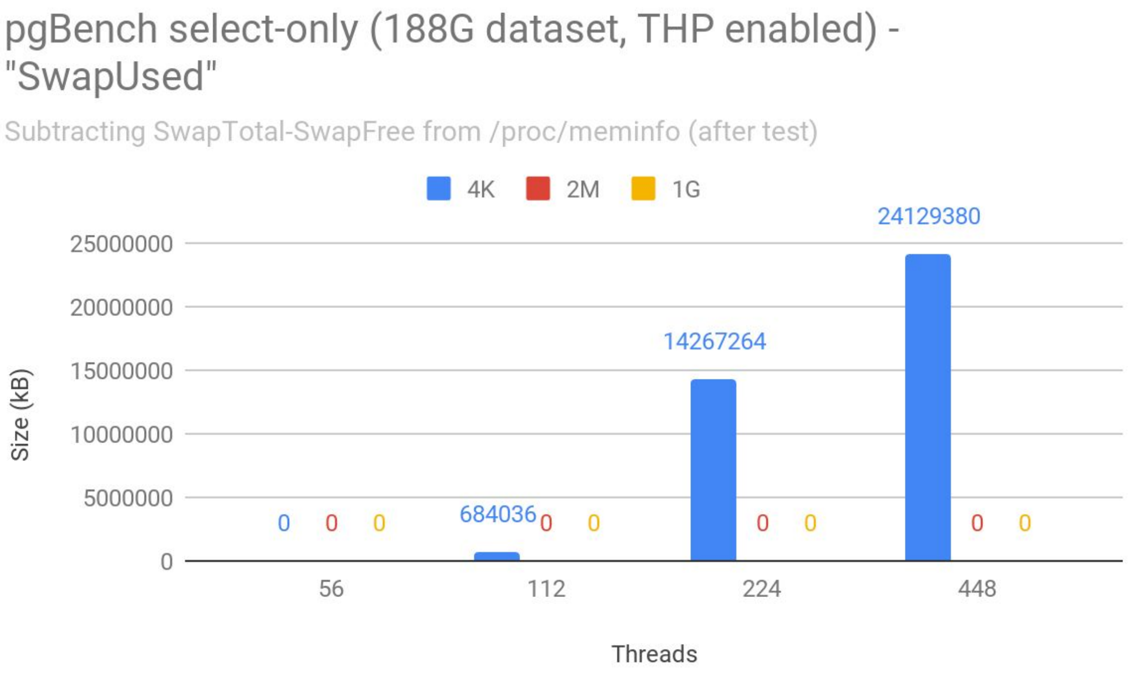
- 4K-pages, 188G shared_buffers, 224 clients

```
top - 19:11:45 up 2 days, 22:42,  3 users,  load average: 271.75, 268.84, 247.22
Tasks: 838 total,   2 running, 835 sleeping,   0 stopped,   1 zombie
%Cpu(s):  0.9 us,  0.6 sy,  0.0 ni, 24.1 id, 74.3 wa,  0.0 hi,  0.1 si,  0.0 st
KiB Mem : 26404166+total,   269488 free, 83409952 used, 18036222+buff/cache
KiB Swap: 58609660 total, 45334072 free, 13275588 used.   855732 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
19749 fernando  20   0  121028  10248   2408 R  27.3  0.0  27:03.98 pgbench
  308 root      20   0       0      0      0 S  22.4  0.0  12:41.40 kswapd1
  307 root      20   0       0      0      0 S  10.1  0.0  22:52.80 kswapd0
```

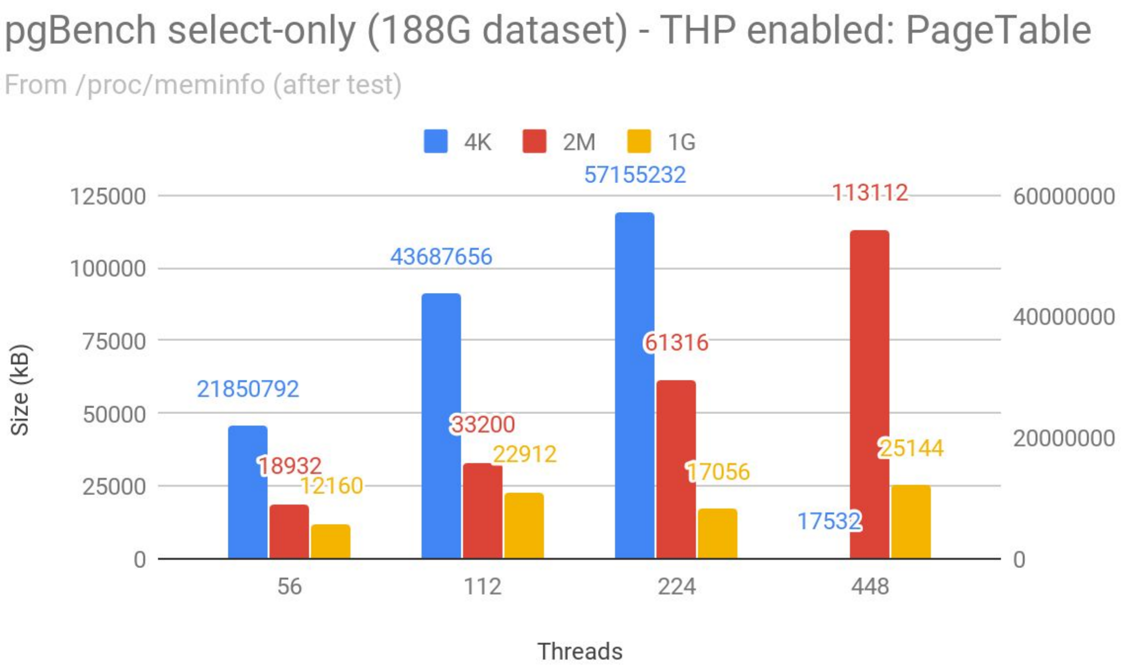

PERCONA

pgBench select-only: PostgreSQL



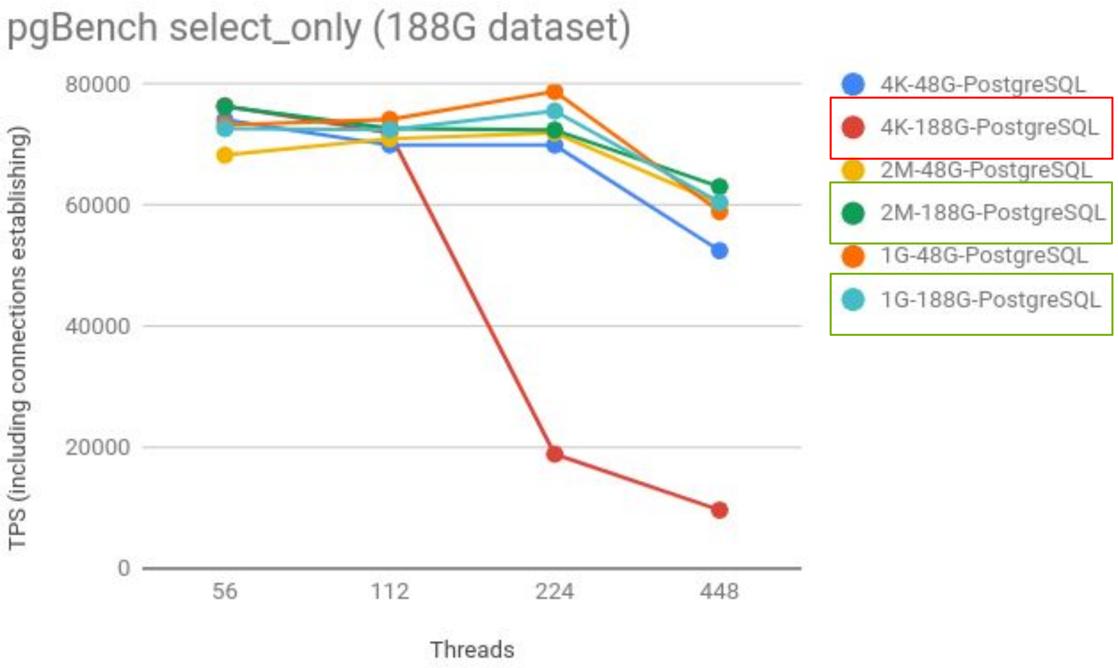
PERCONA

pgBench select-only: PostgreSQL



PERCONA

pgBench select-only: PostgreSQL



Static hugepages cannot be swapped out

PERCONA

What I have learnt

Sharing my findings

Parting thoughts

- It was a much bigger adventure than I anticipated
- The overall idea that databases will greatly benefit from huge pages won't always apply
  - I should (and will) explore a broader range of benchmarks to better understand what types of workloads most benefit from it
- MySQL support for 1G huge pages need some work
  - memory allocation during BP initialization is particular with 1G HP
- Huge pages and swapping

PERCONA

Champions of Unbiased
Open Source Database Solutions