Как выжать максимум производительности из операционной системы и hardware

Илья Космодемьянский
ik@postgresql-consulting.com

PostgreSQL-Consulting.com

Что мы будем знать в результате этого семинара?

- Что такое хороший сервер для базы данных PostgreSQL?
- Как не сделать из хорошего сервера плохой?
- Как сделать из хорошего сервера хороший сервер?

Чего мы всегда хотим?

- Производительность
- Надежность
- Доступная цена

Чего мы всегда хотим?

- Производительность
- Надежность
- Доступная цена
- Чем не теорема? Уж точно не хуже CAP

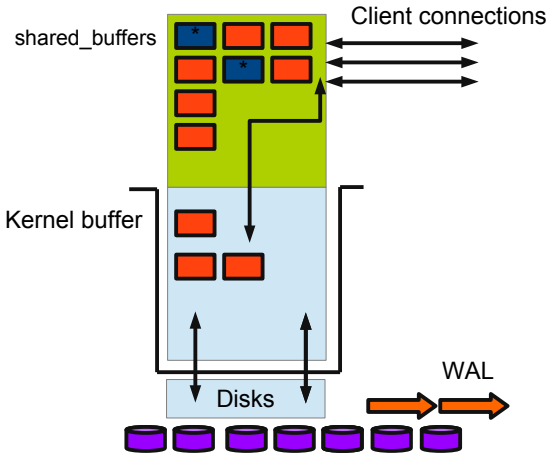
Итак сформулируем критерии

- Хорошие характеристики: CPU, RAM, диски
- Хороший баланс этих характеристик
- Серверное железо, не десктоп под видом сервера

Производительность

- Время отклика
- Складывается из ожидания разных ресурсов
- Простой по причине ненадежности железа увеличивает среднее время отклика

Что особенного в PostgreSQL-специфичном workload



Для чего и как PostgreSQL использует память

- shared_buffers
  - "Страничный кэш"
  - "Чистые" страницы "поднимаются" в shared_buffers, если хоть один tuple изменился, страница помечается как "грязная"
  - COMMIT; не возвращает управления, пока страница не записана в WAL
  - Время от времени происходит CHECKPOINT: грязные страницы из shared_buffers пишутся на диск (fsync)
- work_mem и ее разновидности
  - Если отдельному процессу нужна память (сортировки, хэширование, конкурентное перестроение индекса)

Выбор сервера

- "Большой ящик" vs "много маленьких"
- Платформа
- CPU
- RAM
- Дисковая подсистема
- Операционная система
- Профиль нагрузки

Большой ящик почти всегда лучше

- Если вы "помещаетесь" в один "большой ящик" вам повезло
- Бизнес всегда плохо выделяет деньги под ту нагрузку, которую отдел маркетинга обещает обеспечить
- Преждевременный шардинг это почти всегда плохо

Платформа это о надежности и расширяемости

- Бренд это не всегда хорошо, но почти всегда лучше чем совсем noname
- Сколько еще памяти вы сможете вставить в ваш сервер и чего будет стоить сделать это через три года
- Лучшая платформа - та, которой доверяет ваш админ

CPU для базы не важен

- Это не так
- У PostgreSQL процессная модель
- При интенсивной нагрузке на 8-ядерном сервере ваши дефолтные max_connections=100 будут заниматься ожиданием друг друга
- Есть CPU-емкие задачи и у базы данных
- То есть рекоммендация - выбирать характеристики CPU исходя из желаемомго параллелизма

RAM

- Много не бывает
  - Память быстрее диска
  - Стоит адекватных денег
- Может не эффективно расходоваться

shared_buffers # postgresql.conf

- 25% RAM - хорошая отправная точка
- 75% - может быть хорошо если база помещается в память
- большие объемы shared_buffers - только при хороших дисках

shared_buffers # postgresql.conf

Проблемы

- Много памяти
- Мало памяти
- SWAP vm.swappiness = 1 #sysctl, 0 - плохо ибо OOM-killer
- NUMA
- Плохие диски
- Ожидание CPU

Если памяти много

Huge pages - что это такое?

- Сервера с большим количеством памяти (128-256Gb) теперь стоят разумных денег
- Держать базу в памяти хорошо (любую хорошо, PostgreSQL совсем хорошо)
- По умолчанию память выделяется по 4kB - так быстрее
- ОС транслирует виртуальные адреса в физические, результат кэшируется в Translation Lookaside Buffer (TLB)
- $\frac{1Gb}{4kB} = 262144$ и 64-битная адресация - на больших объемах памяти оверхэд и низкая эффективность TLB
- Выделяем память большими порциями

Если памяти много

Как заставить PostgreSQL использовать huge pages

- *vm.nr_hugepages* = 3170 (sysctl, ядро должно поддерживать)
- До 9.2 включительно - через библиотеку hugetlb
- 9.3 улучшена работа с shared memory с помощью mmap - huge pages не работают
- 9.4+ *huge_pages* = *try*|*on*|*off* (postgresql.conf)
- *try* - дефолт, не получилось, не используем
- Сейчас работат только на linux, *try* на FreeBSD - OK

Мало памяти

- Если памяти не хватает на нормальный размер shared_buffers, все будет медленно
- Минимальный (нормальный) размер определяется размерами базы, характером нагрузки и пропускной способностью всего стэка
- Есть более хитрые способы
- Если не настроен автовакуум, памяти не хватит никогда

Более хитрые способы

http://www.keithf4.com/a-large-database-does-not-mean-large-shared_buffers/

- Базовая идея: большой размер базы не означает автоматически большой размер shared_buffers
- **Проверять с помощью pg_buffercache что-либо надо осторожно!**
- Причин, по которой shared_buffers используются неэффективно может быть больше одной
- Ограничение shared_buffers в 8Gb - это апокриф. (Есть еще Windows)

NUMA

Как понять, что что-то пошло не так?

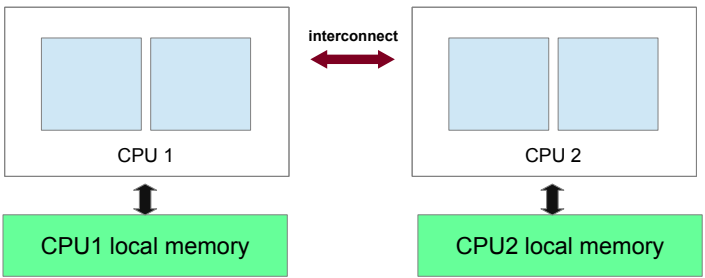
- Некоторые CPU перегружены без очевидных причин

NUMA

Что происходит?

- NUMA-архитектура: Non Uniform Memory Access
- Блок CPU + RAM, такие блоки соеденены друг с другом интерконнектом
- CPU сначала использует свою собственную память, дальше добирает через интерконнект (*numactl – – hardware можно оттрассировать расстояние*)
- Если interleaving → disabled (эффективно означает NUMA → on), CPU отдает приоритет использованию своей локальной памяти (например для shared_buffers;-))

NUMA


interconnect
CPU 1
CPU 2
CPU1 local memory
CPU2 local memory

NUMA

Какие настройки NUMA лучше для PostgreSQL

- Отключить NUMA на низком уровне:
  - Memory interleaving → enable в BIOS
  - numa → off в ядре при загрузке
- Другой вариант:
  - *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

NUMA

NUMA - итого

- Выключенный режим NUMA - хорошо для PostgreSQL (для серверов приложений или например гипервизоров виртуальных машин - обычно наоборот)
- Memory interleaving → enable означает **выключенный** режим NUMA

Плохие диски (в контексте shared_buffers)

- Нет способов заставить работать быстро сервер с нагруженной OLTP-базой, 256Gb памяти и одним SATA-диском

Дисковая производительность

- Традиционно проблемное место для любой базы данных
- Традиционно проблемы возникают:
  - При записи WAL
  - Когда случается CHECKPOINT
  - avtovacuum, pg_clog, tmp, sorts, hashing etc

Checkpoint

Зачем нужны чекпойнты?

- Более быстрое восстановление: undo и redo нужно делать только до чекпойнта, а не по всему WAL
- Но при больших shared_buffers CHECKPOINT запросто может перегрузить дисковую подсистему

Checkpoint

Диагностика

- Скачки дисковой утилизации на **графиках** (iostat -d -x 1, последняя колонка *%util*)
- *pg_stat_bgwriter*

Мониторинг

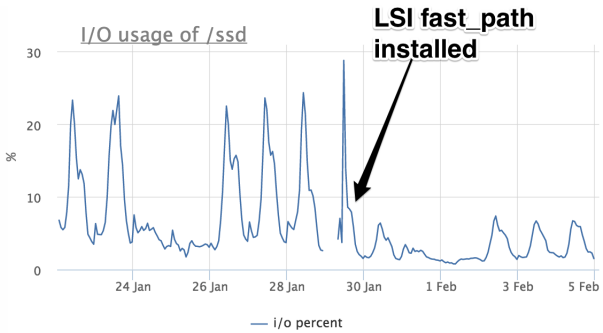
Как минимум

- IOPS - примерно бесполезно, если это единственная метрика
- % utilization
- latency

Хорошо бы иметь

- iowait
- Mbps

График позволяет увидеть трэнд!



pg_stat_bgwriter

```
pgbench=# select * from pg_stat_bgwriter ;
-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 29
checkpoints_req       | 13
checkpoint_write_time | 206345
checkpoint_sync_time  | 9989
buffers_checkpoint    | 67720
buffers_clean         | 1046
maxwritten_clean      | 0
buffers_backend       | 48142
buffers_backend_fsync | 0
buffers_alloc         | 30137
stats_reset           | 2014-10-24 17:59:15.812002-04

postgres=# select pg_stat_reset_shared('bgwriter');
-[ RECORD 1 ]--------+-
pg_stat_reset_shared |
```

pg_stat_bgwriter

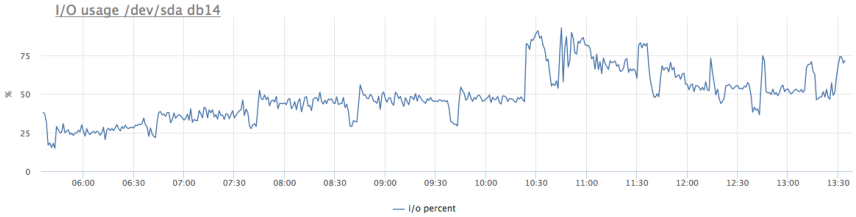
```
pgbench=# select * from pg_stat_bgwriter ;
-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 29
checkpoints_req       | 13
checkpoint_write_time | 206345
checkpoint_sync_time  | 9989
buffers_checkpoint    | 67720
buffers_clean         | 1046
maxwritten_clean      | 0
buffers_backend       | 48142
buffers_backend_fsync | 0
buffers_alloc         | 30137
stats_reset           | 2014-10-24 17:59:15.812002-04

postgres=# select pg_stat_reset_shared('bgwriter');
-[ RECORD 1 ]--------+-
pg_stat_reset_shared |
```

Это неправильный pg_stat_bgwriter

pg_stat_bgwriter - немного получше

```
postgres=# select *, now() from pg_stat_bgwriter ;
-[ RECORD 1 ]---------+------------------------------
checkpoints_timed     | 0
checkpoints_req       | 38
checkpoint_write_time | 20288693
checkpoint_sync_time  | 34751
buffers_checkpoint    | 9176173
buffers_clean         | 0
maxwritten_clean      | 0
buffers_backend       | 10521857
buffers_backend_fsync | 0
buffers_alloc         | 9815168
stats_reset           | 2015-03-22 06:00:02.601286+03
now                   | 2015-03-22 16:01:21.3482+03
```

Еще лучше иметь графики на одной странице



L-Consulting.com

Что делать?

Hardware: RAID

- RAID контроллер должен быть с "батарейкой"
- **Но!** при использовании хороших enterprise-level SSD кэш на диске лучше кэша на контроллере!
- Производители: megaraid или perc - OK, а например HP или ARECA - не всегда
- Батарейка - присутствует и исправна
- cache mode → write back (т.к. у нас есть батарейка)
- io mode (ip policy) → direct (у PostgreSQL свой кэш)
- Disk Write Cache Mode → disabled (с SSD есть тонкости!)

Что делать?

Hardware: Диски

- 2,5"SAS (вполне бывает 15K) - seek в 2-3 раза быстрее чем 3,5"
- Не все SSD одинаково полезны: enterprise level Intel p3700 vs desktop-level Samsung
- Использовать только SSD для PostgreSQL означает быть готовыми к ряду проблем - **устаревший тезис**
- В настоящий момент нет причин НЕ держать OLTP-базу на SSD. Можете себе позволить - используйте!
- RAID 1+0
- Если нет хороших дисков или контроллера - *synchronous_commit* $\rightarrow$ *off*

Что делать?

Hardware: SSD

- NAND vs DRAM
- NAND
  - Ограничено кол-во циклов
  - SLC, MLC
  - конденсатор
- random_page_cost должен быть ближе к seq_page_cost #postgresql.conf
- effective_io_concurrency лучше увеличить #postgresql.conf
- FusionIO, Verident etc - всё вышесказанное справделиво в еще большей степени

Что делать?

Файловая система

- xfs или ext4 - OK
- barrier=0, noatime (барьер нельзя отключать с плохими SSD и без BBU)
- ZFS
  - JBOD vs RAID-Z
  - За удобство надо платить производительностью

Что делать?

Операционная система

- По умолчанию $vm.dirty\_ratio = 20$ $vm.dirty\_background\_ratio = 10$
- Иными словами pdflush начинает сбрасывать грязные страницы только когда их 10% от всей памяти в сервере
- Более разумные значения $vm.dirty\_background\_bytes = 67108864$ $vm.dirty\_bytes = 536870912$ (512Mb кэш на RAID)
- Если нет кэша на RAID - разделить на 4

Что делать?

postgresql.conf

- *wal_buffers* (768kB → 16Mb)
- *checkpoint_segments* (3 - чекпойнт каждые 48Mb → 256 - 4Gb)
- Начиная с 9.5 стало проще *checkpoint_segments* заменен на *min_wal_size* и *max_wal_size*
- *checkpoint_timeout* = 60 (что первое наступит)
- *checkpoint_completion_target* = 0.9 (для распределения чекпойнтов)

Как себя проверить?

```
pgdev@pg-dev-deb:~$ tt_pg/bin/pg_test_fsync
5 seconds per test
O_DIRECT supported on this platform for open_datasync and open_sync.

Compare file sync methods using one 8kB write:
(in wal_sync_method preference order, except fdatasync
is Linux's default)
        open_datasync                  11396.056 ops/sec      88 usecs/op
        fdatasync                      11054.894 ops/sec      90 usecs/op
        fsync                          10692.608 ops/sec      94 usecs/op
        fsync_writethrough                           n/a
        open_sync                         67.045 ops/sec   14915 usecs/op

Compare file sync methods using two 8kB writes:
(in wal_sync_method preference order, except fdatasync
```

Небольшая хитрость: пусть bgwriter работает

```
postgres=# select name, setting, context, max_val, min_val  from pg_settings where name ~ 'bgwr';
  name            | setting | context | max_val | min_val
--------------------------+---------+---------+---------+---------
 bgwriter_delay           | 200     | sighup  | 10000   | 10
 bgwriter_lru_maxpages    | 100     | sighup  | 1000    | 0
 bgwriter_lru_multiplier | 2       | sighup  | 10      | 0
(3 rows)
```

*bgwriter спит (как правило!) bgwriter_delay ms, затем списывает на диск столько же буферов, сколько понадобилось чистых в прошлый проход * bgwriter_lru_multiplier, но не больше bgwriter_lru_maxpages за один проход*

Операционная система

- Linux
- Новые ядра
- Производительность и testbase - главный аргумент против FreeBSD
- HP UX и Solaris - в настоящий момент странно, если нет специальных аргументов
- Windows + PostgreSQL = много проблем

Linux-специфичная настройка: шедуллер

- kernel.sched_migration_cost_ns - время, в течение которого шедуллер считает процасс достаточно горячим, чтобы не мигрировать его на другой CPU
- kernel.sched_autogroup_enabled - шедуллер группирует процессы от одного TTY на один CPU

Проверить версию ядра!

Пример настройки шедуллера

```
$ 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 в большинстве случаев будет изрядно быстрее acpi_cpufreq + ondemand
- intel_pstate + powersave

Linux-специфичная настройка: pg_bouncer

- net.ipv4.ip_local_port_range - может понадобится увеличить
- limits (важно не только для pg_bouncer)

Виртуализация

- IO было и остается главной проблемой
- основная проблема IO это latency
- Производительный DBaaS дороже анлогичного сервера (иногда включая его обслуживание)

Вопросы?

- На нашей стойке PGDay'16 Russia - сегодня, завтра и послезавтра
- **ik@postgresql-consulting.com**