

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

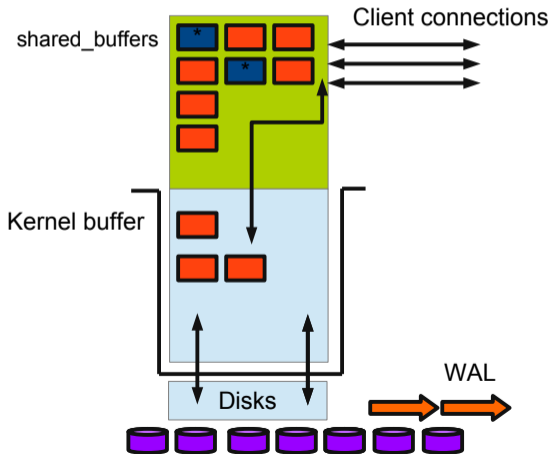


Чего мы всегда хотим?

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

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

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



- `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)

- "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 понаме
- 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 рекомендация - выбирать характеристики CPU исходя из желаемого параллелизма

- 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

Проблемы

- Много памяти
- Мало памяти
- SWAP `vm.swappiness = 0 #sysctl`
- NUMA
- Плохие диски
- Ожидание CPU

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

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 работают только на linux, `try` на FreeBSD - ОК

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

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)

- Пороявляется обычно в нагрузке не очень понятного происхождения на отдельные ядра
- numa → off или `vm.zone_reclaim_mode = 0`
- То есть по уму лучше `numactl --interleave = all /etc/init.d/postgresql start`
- Ещё лучше чтобы база разруливала сама, но это светлое будущее

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

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

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

Диагностика

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

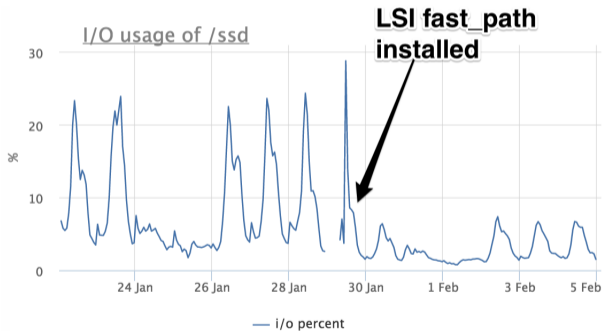
Как минимум

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

Хорошо бы иметь

- iowait
- Mbps

График позволяет увидеть тренд!





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 |
```

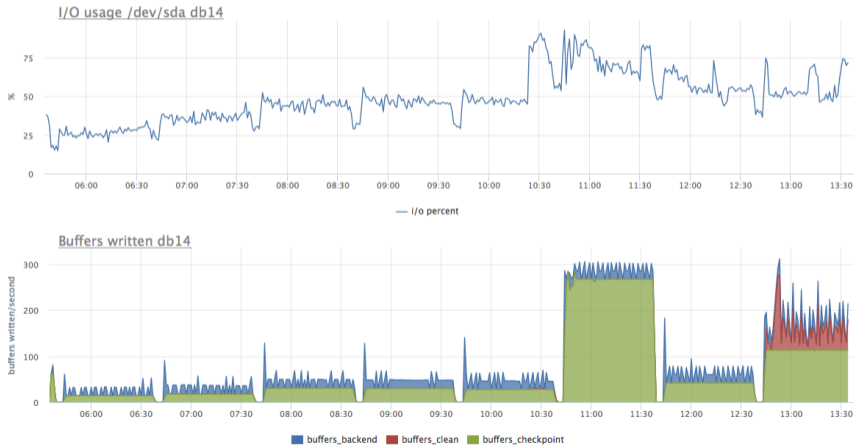
Это неправильный `pg_stat_bgwriter`



pg_stat_bgwriter - немного лучше

```
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



Hardware: RAID

- Дешевый RAID контроллер (без чипа) - зло, уж лучше software RAID
- RAID контроллер должен быть с "батарейкой"
- Производители: megaraid или perc - ОК, 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!)

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
- В настоящий момент нет причин НЕ держать OLTP-базу на SSD
- RAID 1+0
- больше "шпинделей"
- Если нет хороших дисков или контроллера - *synchronous_commit* → off

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 справедливо в еще большей степени

Файловая система

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

Операционная система

- По умолчанию $vm.dirty_ratio = 20$ $vm.dirty_background_ratio = 10$ - плохо
- Более разумные значения $vm.dirty_background_bytes = 67108864$
 $vm.dirty_bytes = 536870912$ (512Mb кэш на RAID)
- Если нет кэша на RAID - разделить на 4

postgresql.conf

- *wal_buffers* (768kB → 16Mb)
- *checkpoint_segments* (3 - чекпойнт каждые 48Mb → 256 - 4Gb)
- *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 fdatsync is Linux's default)

open_datasync	11396.056 ops/sec	88 usecs/op
fdatsync	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 fdatsync

```
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*

- 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

- `kernel.sched_migration_cost_ns` - если процессов много, более высокие значения могут помочь
- `kernel.sched_autogroup_enabled` - шедуллер лучше группирует процессы от одного псевдоюзера по CPU

Проверить версию ядра!

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

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



Вопросы?

ik@postgresql-consulting.com

PostgreSQL-Consulting.com