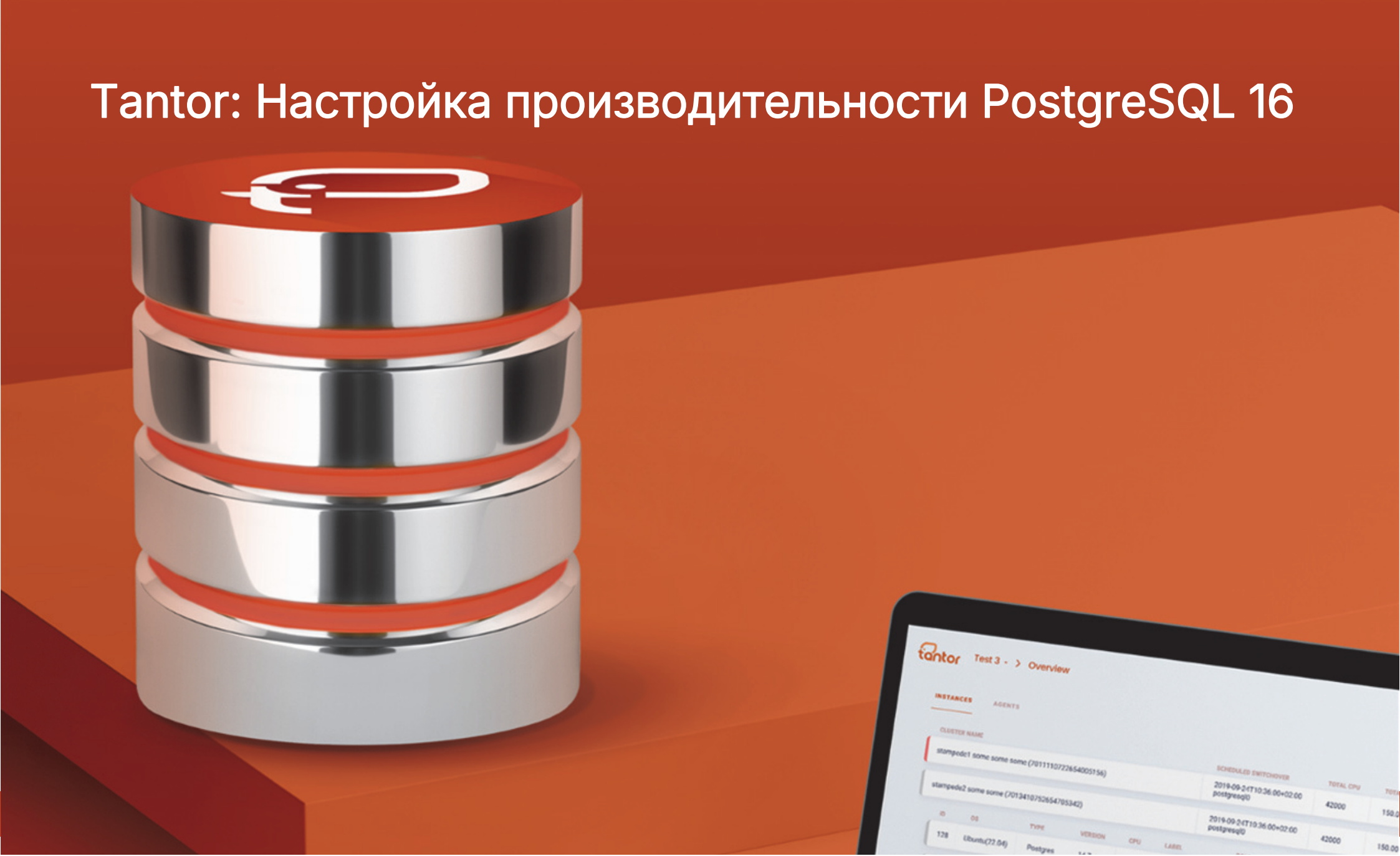
Tantor: Настройка производительности PostgreSQL 16

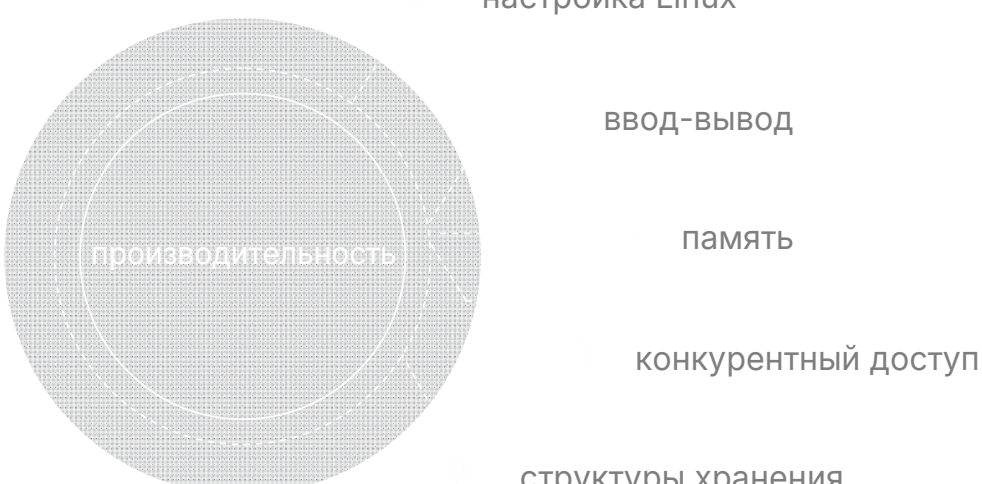


Введение

Настройка производительности

Обзор

- в курсе изучаются:
  - настройка операционной системы Linux
  - настройка экземпляра, обслуживающего кластер баз данных
  - оптимизация структур хранения
- в курсе не изучается:
  - настройка кода SQL
  - настройка кластеров Patroni



О компании Тантор

- с 2016 года на международном рынке
- с 2021 года на российском рынке
- разработка СУБД Tantor для государственных и коммерческих организаций
- разработка Платформы Tantor для мониторинга и управления СУБД семейства PostgreSQL, а также кластеров Patroni
- многолетний опыт эксплуатации высоконагруженных систем
- входит в Группу Компаний «Астра»

СУБД Tantor

Tantor BE

Новые возможности и доработки по сравнению с PostgreSQL, техническая поддержка

Tantor SE

СУБД Enterprise-уровня, подходит для наиболее нагруженных OLTP-систем или КХД размером до 100ТБ

Tantor SE 1C

СУБД для высоких нагрузок, оптимизированная и одобренная для работы с приложениями 1С

Tantor PipelineDB

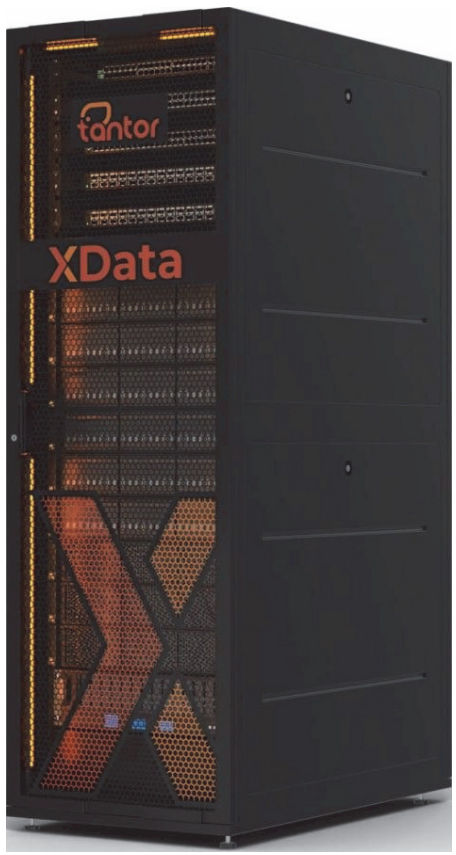
Расширение, позволяющее непрерывно обрабатывать данные

В составе Tantor xData

Максимальная версия СУБД, оптимизированная для работы с 1С

Tantor xData

- программно-аппаратный комплекс, с высокой производительностью, отказоустойчивостью, безопасностью
- возможности DBaaS в ЦОД предприятия
- Улучшенная автоматизация и резервное копирование
- высокая производительность и масштабируемость
- снижение затрат на инфраструктуру и администрирование
- в состав входит СУБД Tantor и Платформа Tantor



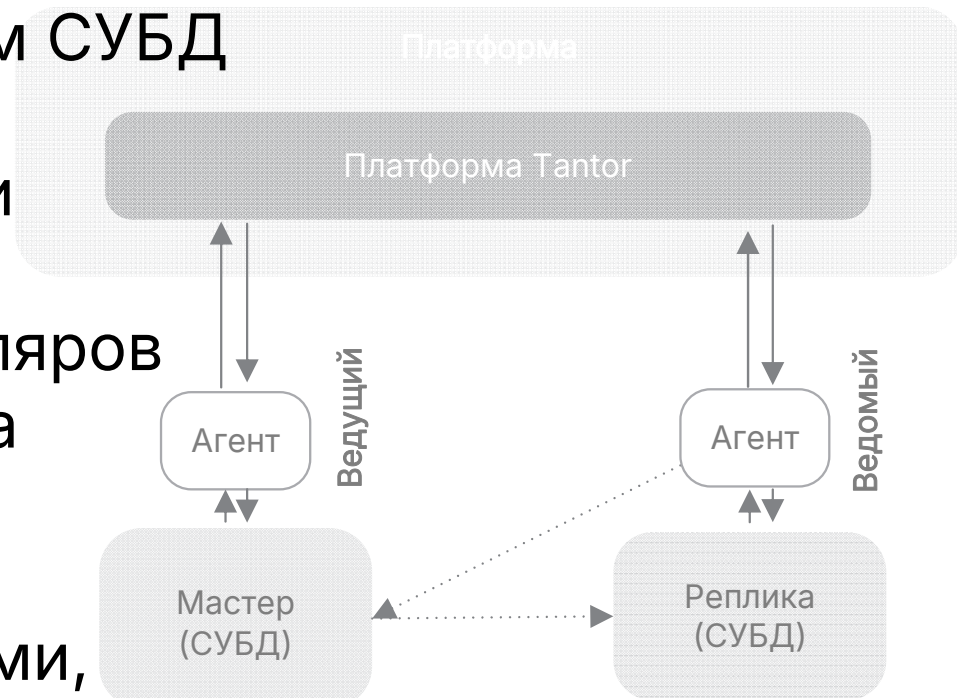
Tantor PipelineDB

- расширение для СУБД Tantor и PostgreSQL с открытым исходным кодом для непрерывного выполнения SQL-запросов на потоках данных с инкрементальным сохранением результатов в обычных таблицах
- высокопроизводительное агрегирование временных рядов
- позволяет соединять потоковые данные с историческими данными для сопоставления в режиме реального времени
- может использоваться в приложениях, где нужна незамедлительная реакция
- пример непрерывного представления для вывода ежесуточного трафика, используемого топ десятью ip-адресов:

```
CREATE VIEW heavy_hitters AS
 SELECT day(arrival_timestamp), topk_agg(ip, 10, response_size)
 FROM requests_stream GROUP BY day
```

Платформа Tantor

- программное обеспечение для управления большим количеством СУБД и кластеров Patroni
- управляет СУБД Tantor и форками PostgreSQL
- сбор показателей работы экземпляров PostgreSQL, хранение и обработка показателей, рекомендации по настройке производительности
- интеграция с почтовыми системами, службами каталогов, мессенджерами



О курсе

- очное или дистанционное обучение с инструктором:
  - продолжительность 5 дней
  - начало в 10:00
  - перерыв на обед 13:00-14:00
  - окончание до 17:00 (последний день до 15:00)

Общие понятия настройки производительности

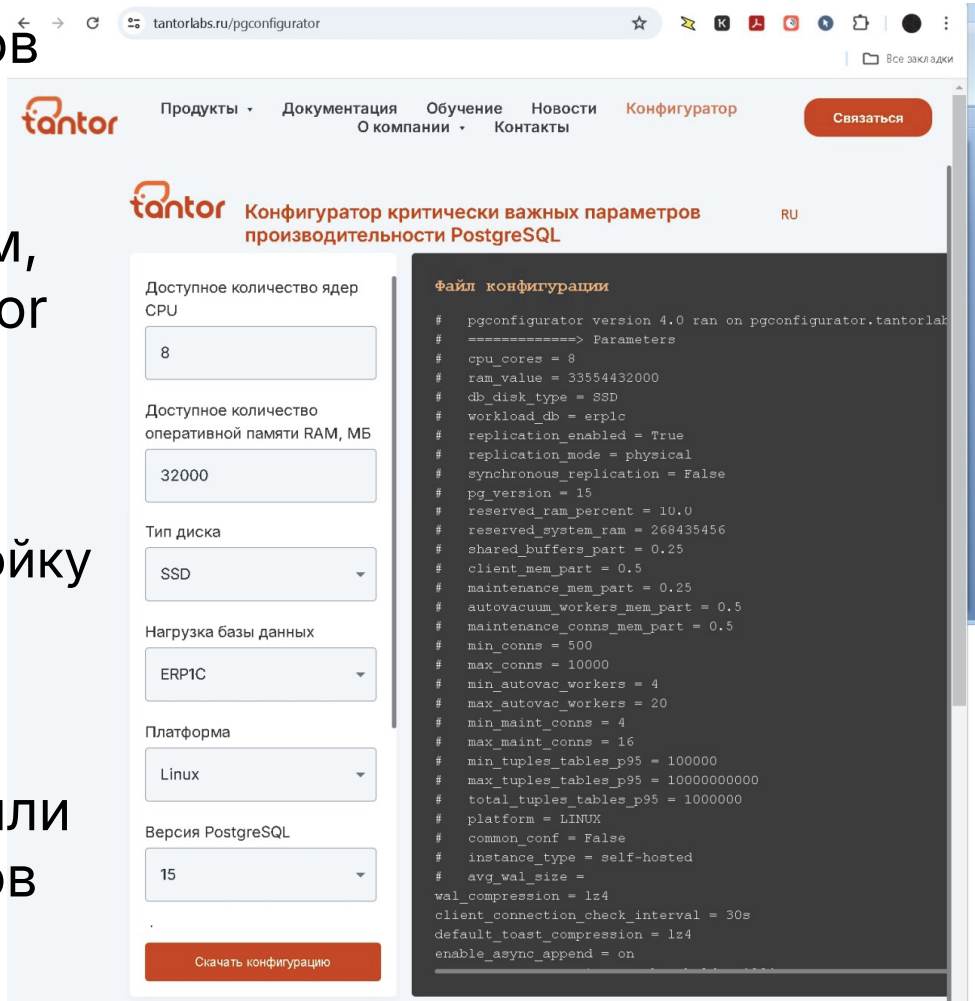
- цели настройки производительности:
  - при вводе в эксплуатацию, разработке, приложения, выборе оборудования
  - миграции с другой СУБД
  - в процессе эксплуатации при обнаружении снижения целевых показателей
- настройка производительности включает в себя:
  - целевые показатели (метрики), которые описывают ожидания пользователей приложений по качеству обслуживания. Показатели могут быть указаны в соглашении об уровне обслуживания
  - процедуры получения показателей (мониторинг)
  - инструменты настройки производительности: утилиты, расширения, функции, командные файлы.

Методология настройки производительности

- не зависит от используемых инструментов
- включает в себя:
  - оценку архитектуры приложения: как приложение взаимодействует с базой данных
- ошибки на уровне архитектуры приложения создают узкие места и определяют ограничения в рамках которых возможна настройка производительности
- если в архитектуру приложения можно внести изменения, устраняющие узкие места, то достигается наибольший эффект

Последовательность действий по настройке производительности

- настройка параметров хранения файлов кластера баз данных
- проверка параметры конфигурации экземпляра на соответствие значениям, рекомендуемым конфигуратором Tantor или Платформы Tantor
  - Конфигуратор Tantor (http://tantorlabs.ru/pgconfigurator) позволяет выполнить начальную настройку параметров конфигурации СУБД PostgreSQL
- после тестовой нагрузки или при эксплуатации приложения проверить или провести тонкую настройку параметров конфигурации экземпляра



Последовательность действий по настройке производительности

- определение целевые показателей, по достижении которых прекращают настройку производительности
- выбор подсистемы экземпляра, усилия по настройке которой дадут наибольший эффект
  - подсистема, которая обслуживает наиболее дефицитный ресурс является узким местом
- измерение показателей производительности, относящиеся к подсистеме
- внесение изменений в параметры конфигурации, относящиеся к подсистеме
  - если показатели улучшились, то изменения значений параметров были успешны
- при достижении целевых показателей настройка завершается

1-2

Использование утилиты pgbench

Бенчмаркинг

- бенчмаркинг - это процесс тестирования производительности оборудования, программного обеспечения или всей системы в целом
  - benchmark - критерий, ориентир
- универсальным показателем является число команд в секунду (transactions per second, tps), которые может обработать СУБД
  - если транзакция состоит из одной команды, то этот показатель называют число запросов в секунду (queries per second, qps)
  - используется для сравнения производительности до внесения изменений в конфигурацию СУБД и после внесения изменений
- для измерения показателя нужны:
  - набор команд, который будет выполняться в транзакции
  - таблицы с данными и другие объекты (индексы, ограничения целостности, последовательности), которые нужны для выполнения команд
  - число сессий, в которых параллельно будут выполняться транзакции

Результат бенчмаркинга

- основное это tps в последней строке
- стандартное отклонение выводится для задержки (latency)
- параметр -P позволяет выдавать текущие значения tps и latency

```
pgbench -T 30 -P 5
starting vacuum...end.
progress: 5.0 s, 582.0 tps, lat 1.709 ms stddev 0.252, 0 failed
progress: 10.0 s, 597.0 tps, lat 1.667 ms stddev 0.199, 0 failed
progress: 15.0 s, 596.0 tps, lat 1.670 ms stddev 0.274, 0 failed
progress: 20.0 s, 581.8 tps, lat 1.712 ms stddev 0.186, 0 failed
progress: 25.0 s, 601.4 tps, lat 1.655 ms stddev 0.206, 0 failed
progress: 30.0 s, 582.4 tps, lat 1.710 ms stddev 0.213, 0 failed
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 1
number of threads: 1
maximum number of tries: 1
duration: 30 s
number of transactions actually processed: 17704
number of failed transactions: 0 (0.000%)
latency average = 1.687 ms
latency stddev = 0.225 ms
initial connection time = 3.788 ms
tps = 590.180430 (without initial connection time)
```

pgbench - утилита бенчмаркинга PostgreSQL

- утилита командной строки, стандартно поставляемая с PostgreSQL
- используется для создания тестовой нагрузки при настройке производительности
- встроенные тесты используют четыре таблицы:
  - `pgbench_accounts` (100тыс.строк), `pgbench_tellers` (10 строк), `pgbench_branches` (1 строка), `pgbench_history` (0 строк)

```
create table pgbench_history (tid int, bid int, aid int, delta int, mtime timestamp);
create table pgbench_tellers (tid int primary key, bid int, tbalance int, filler char(84));
create table pgbench_accounts (aid int primary key, bid int, abalance int, filler char(84));
create table pgbench_branches (bid int primary key, bbalance int, filler char(88));
```

- таблицу `pgbench_accounts` можно сделать секционированной

```
pgbench -i -s 100
dropping old tables...
creating tables...
generating data (client-side)...
10000000 of 10000000 tuples (100%) done (elapsed 39.06 s, remaining 0.00 s))
vacuuming...
creating primary keys...
done in 85.48 s (drop tables 0.01 s, create tables 0.01 s, client-side generate 39.29 s,
 vacuum 10.91 s, primary keys 35.27 s).
```

Три встроенных теста pgbench

- по умолчанию pgbench запускает тест TPC-B из 7 команд в одной транзакции:

```
BEGIN;
UPDATE pgbench_accounts SET abalance = abalance + :delta WHERE aid = :aid;
SELECT abalance FROM pgbench_accounts WHERE aid = :aid;
UPDATE pgbench_tellers SET tbalance = tbalance + :delta WHERE tid = :tid;
UPDATE pgbench_branches SET bbalance = bbalance + :delta WHERE bid = :bid;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (:tid, :bid, :aid, :delta,
 CURRENT_TIMESTAMP);
END;
```

- TPC-B является простым тестом и не соответствует нагрузке, создаваемой типичными приложениями, работающими с СУБД
- список встроенных тестов:

```
pgbench -b list
Available builtin scripts:
      tpcb-like: <builtin: TPC-B (sort of)>
  simple-update: <builtin: simple update>
    select-only: <builtin: select only>
```

- запуск теста select-only:

```
pgbench -b select-only -T 10 -P 3
```

Параметры запуска pgbench

- `-P секунд` задаёт интервал в секундах через который утилита будет выводить строку со статистикой выполнения.
- `-T секунд` длительность теста
- `--protocol=prepared`. В режиме `prepared` утилита pgbench в каждой сессии передает команду один раз (подготавливает) и дальше передает только новые значения параметров и команду на выполнение

```
pgbench -T 10 -P 3
        ...
        progress: 10.0 s, 530.1 tps, lat 1.883 ms stddev 1.558, 0 failed
        progress: 20.0 s, 427.8 tps, lat 2.319 ms stddev 7.186, 0 failed
        progress: 30.0 s, 174.9 tps, lat 5.753 ms stddev 25.774, 0 failed
        ...
        latency average = 2.645 ms
        latency stddev = 11.181 ms
        ...
        tps = 377.649988 (without initial connection time)
```

Рекомендации по использованию pgbench

- `-c N` задаёт число параллельных сессий, в которых будут выполняться транзакции или скрипты. Сессии создаёт один процесс pgbench по умолчанию одним потоком. Параметром `-j N` можно указать количество потоков (threads)
- число сессий должно быть не меньше числа потоков
- при использовании `-c N` со встроенными тестами, таблицы желательно создать с параметром `-s M`, чтобы `M` было не меньше, чем N
- параметром `-f script.sql@10` можно посылать на выполнение собственный набор команд, сохранённый в файле. В этом случае "транзакцией" в отчёте pgbench будет считаться выполнение всех команд в файле

```
pgbench -T 10 -P 5 -s 3 -c 50 -j 80
pgbench: warning: scale option ignored, using count from pgbench_branches table (100)
scaling factor: 100
number of clients: 50
number of threads: 50
```

Пример использования pgbench

- задача: проверить, что быстрее `count(*)`, `count(1)`, `count(pk)`
- создание таблицы с данными:

```
drop table if exists t;
create table t(pk bigserial, c1 text default 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa');
insert into t select *, 'a' from generate_series(1, 100000);
alter table t add constraint pk primary key (pk);
```

- создание скриптов:

```
echo "select count(*) from t;" > count1.sql
echo "select count(1) from t;" > count2.sql
echo "select count(pk) from t;" > count3.sql
```

- запуск тестов:

```
pgbench -T 30 -f count1.sql 2> /dev/null | grep tps
tps = 74
pgbench -T 30 -f count2.sql 2> /dev/null | grep tps
tps = 66
pgbench -T 30 -f count3.sql 2> /dev/null | grep tps
tps = 54
```

- результат: `count(1)` быстрее `count(pk)` на ~18%, `count(*)` быстрее `count(1)` на ~10%

1-3

Использование утилит sysbench и fio

sysbench - утилита тестирования производительности

- в тестировании команд SQL функционал схож с pgbench
- тестирование выполняется с использованием простой таблицы:

```
\d sbtest1
                                Table "public.sbtest1"
 Column |      Type      | Collation | Nullable |                Default
--------+----------------+-----------+----------+---------------------------------------
 id     | integer        |           | not null | nextval('sbtest1_id_seq'::regclass)
 k      | integer        |           | not null | 0
 c      | character(120) |           | not null | ''::bpchar
 pad    | character(60)  |           | not null | ''::bpchar
Indexes:
    "sbtest1_pkey" PRIMARY KEY, btree (id)
    "k_1" btree (k)
```

- имеет несколько скриптов для тестирования:

```
ls /usr/share/sysbench
bulk_insert.lua  oltp_insert.lua        oltp_read_write.lua        oltp_write_only.lua
oltp_common.lua  oltp_point_select.lua  oltp_update_index.lua     select_random_points.lua
oltp_delete.lua  oltp_read_only.lua     oltp_update_non_index.lua  select_random_ranges.lua
sysbench --db-driver=pgsql --pgsql-port=5432 --pgsql-db=postgres --pgsql-user=postgres --
pgsql-password=postgres --table_size=100000 /usr/share/sysbench/oltp_read_only.lua prepare
```

Использование sysbench для тестирования процессоров, памяти, дисков

- встроенные тесты: cpu, memory, fileio, threads, mutex
- тестирование скорости работы нескольких потоков:

```
sysbench cpu run --time=10 --threads=4 | grep sec
    events per second:  4709.35
sysbench cpu run --time=10 --threads=8 | grep sec
    events per second:  4684.77
```

- оптимальное значение --num-threads число ядер центральных процессоров
- скорость работы (чтение или запись) с памятью:

```
sysbench memory run --memory-block-size=4K --time=10 --memory-oper=read --memory-access-mode=sequential
--memory-scope=local --threads=2 | grep transf
5673.72 MiB transferred (567.27 MiB/sec)
sysbench memory run --memory-block-size=2M ...
102400.00 MiB transferred (38767.60 MiB/sec)
```

Тестирование аппаратных ресурсов

- запуск потоков и их одновременная работа:

```
sysbench threads --threads=128 --time=10 run | grep events:
    total number of events:              1502
sysbench threads --threads=8 --time=10 run | grep events:
    total number of events:              3991
sysbench threads --threads=4 --time=10 run | grep events:
    total number of events:              8278
sysbench threads --threads=1 --time=10 run | grep events:
    total number of events:              2773
```

- тест работы с файлами:

```
sysbench fileio --file-total-size=128M --file-num=8 prepare
134217728 bytes written in 1.19 seconds (107.12 MiB/sec).
sysbench fileio --file-block-size=8K --file-fsync-mode=fdatasync --file-fsync-end=on --file-total-
size=128M --file-num=8 --file-test-mode=rndrw --max-time=10 run | grep /s
    reads/s:                      435.92
    writes/s:                     290.61
    fsyncs/s:                     939.62
    read, MiB/s:                  6.81
    written, MiB/s:               4.54
    events (avg/stddev):          16613.0000/0.00
    execution time (avg/stddev):  9.8619/0.00
```

- тест использования легковесных блокировок (mutex)

```
sysbench mutex run --mutex-num=9000000 | grep exec
    execution time (avg/stddev):   0.1511/0.00
```

Тестирование ввода-вывода утилитой Flexible IO Tester (fio)

- `sysbench` использует тестовые файлы, заполненные нулями
- для тестирования ввода-вывода обычно используют утилиту `fio`
- основные параметры при тестировании оборудования для PostgreSQL:
  - `bs=8k` PostgreSQL по умолчанию читает и пишет блоками по 8Кб
  - `direct=0` PostgreSQL по умолчанию работает с файлами через страничный кэш

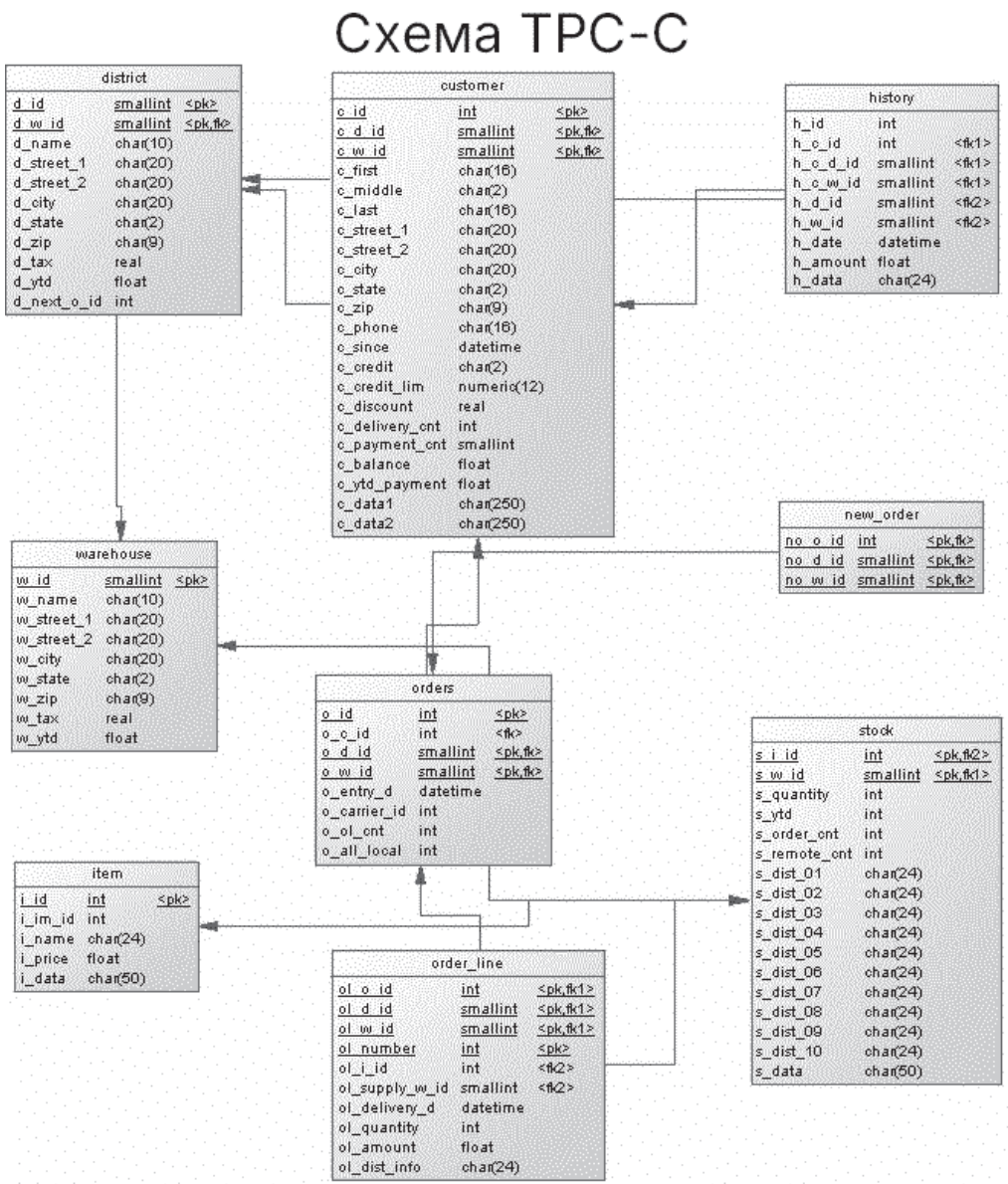
```
sudo apt update && apt install fio -y
sudo fio --rw=rw --rwmixread=75 --size=16m --directory=/ --fadvise_hint=0 --blocksize=8k --
direct=0 --numjobs=1 --nrfiles=1 --runtime=5s --time_based --exec_prerun="echo 3 >
/proc/sys/vm/drop_caches" --group_reporting --name=test1
...
  read: IOPS=15.2k, BW=119MiB/s (124MB/s)(594MiB/5011msec)
     lat (usec): min=3, max=51909, avg=53.39, stdev=860.06
    bw (  KiB/s): min=110946, max=147129, per=100.00%, avg=121645.70, stdev=12274.53, samples=10
    iops        : min=13868, max=18391, avg=15205.60, stdev=1534.21, samples=10
  write: IOPS=5045, BW=39.4MiB/s (41.3MB/s)(198MiB/5011msec); 0 zone resets
     lat (usec): min=4, max=137, avg= 8.57, stdev= 4.52
    bw (  KiB/s): min=37184, max=48542, per=100.00%, avg=40428.10, stdev=3900.63, samples=10
    iops        : min= 4648, max= 6067, avg=5053.30, stdev=487.36, samples=10
  cpu           : usr=23.27%, sys=5.33%, ctx=2279, majf=0, minf=14
...
```

1-4

Тесты TPC

Тесты TPC-B и TPC-C

- в TPC-B команды выполняются на стороне базы данных с максимальной скоростью
- в тесте TPC-B эмулируется работа банка
- в тесте TPC-C эмулируется работа оптового склада
- время выполнения транзакций в TPC-C 5 секунд для всех транзакций кроме одной где время выполнения до 20 секунд
- основной результат это tpmC - число транзакций в минуту

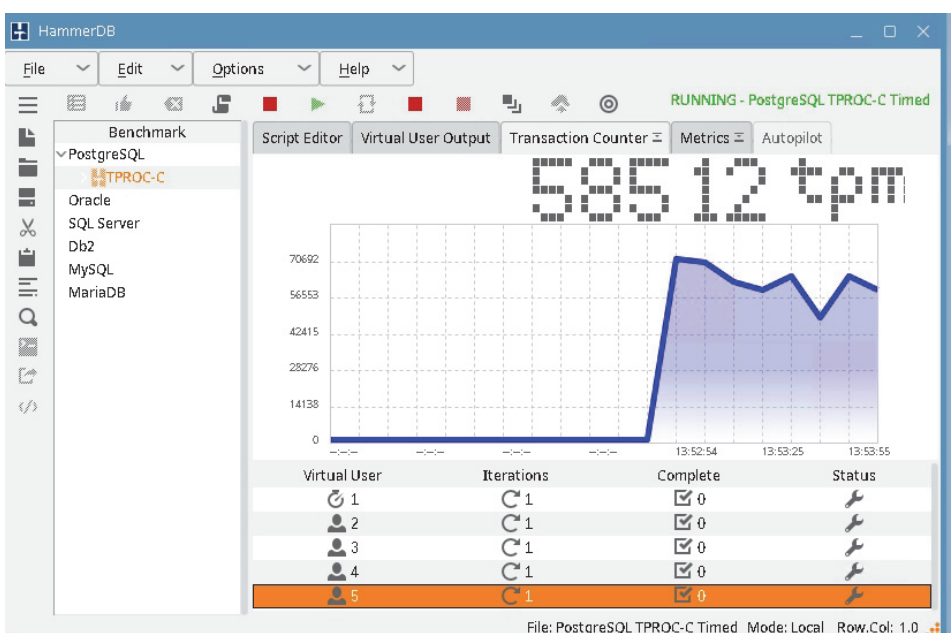


Тест TPC-E, тестирование устойчивости к сетевым сбоям

- TPC-E тест для OLTP, появился в 2006 году
- вместо оптового склада описывается работа брокерской компании, количество таблиц 33 (TPC-E), вместо 9 (TPC-C)
- первичных ключей 33 вместо 8, внешних ключей 50 вместо 9
- из-за сложности воспроизведения и реализации тестов TPC востребованы простые утилиты
- для баз данных семейства PostgreSQL используется утилита pgbench

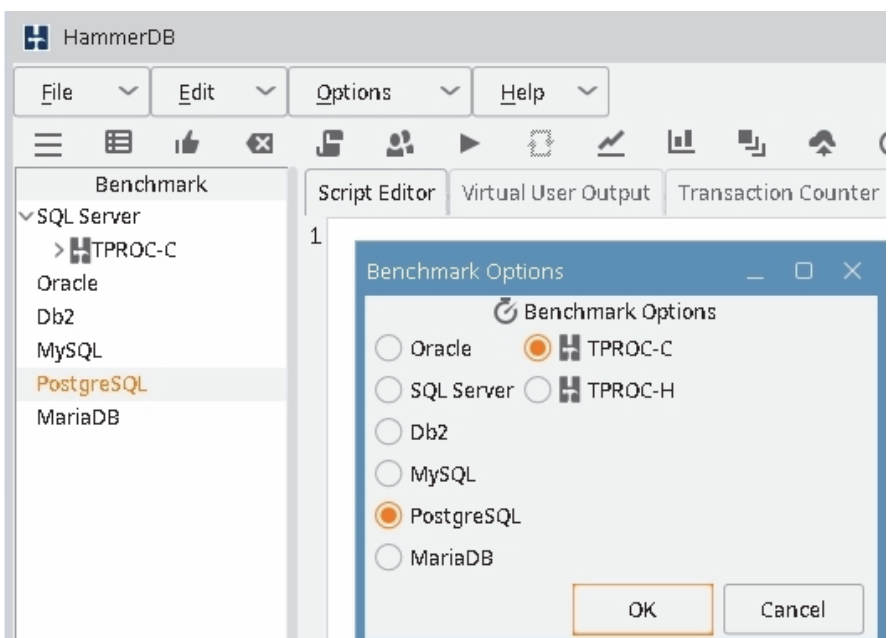
Реализация теста TPC-C

- использует в качестве результата измерений tpmC число "транзакций" этого теста в минуту
- может использоваться для сравнения производительности СУБД разных производителей
- для оценки влияния изменений конфигурации экземпляра не удобен
- недостаток теста в переусложнении правил теста, что приводит к увеличению объема данных в таблицах, количества клиентов и памяти, которые они потребляют
- использует набор команд SQL на порядок сложнее, чем в TPC-B и использует 9 таблиц
- приложение HammerDB включает в себя варианты тестов TPC-C и TPC-H



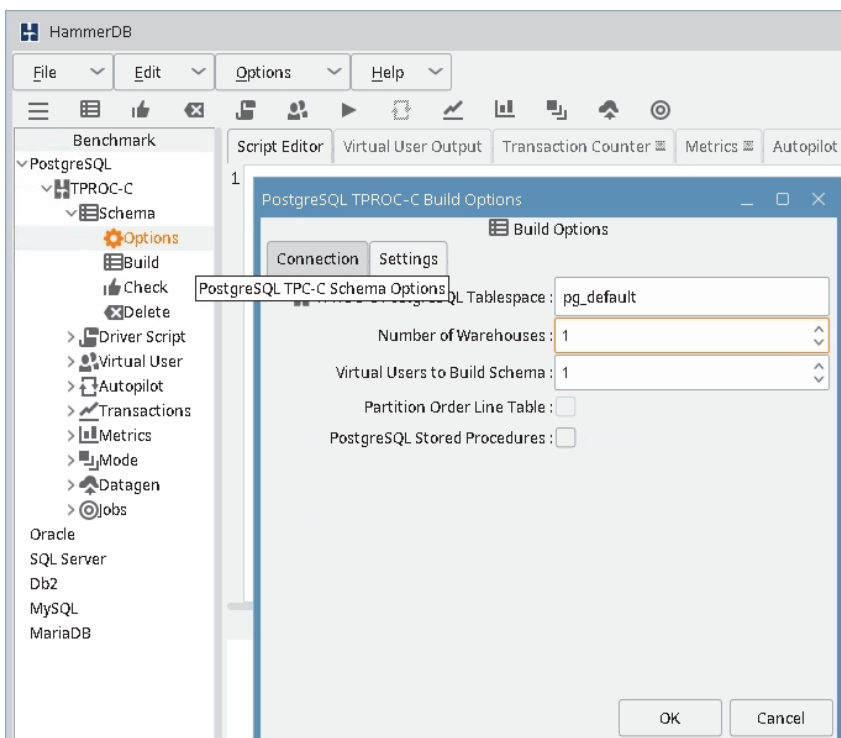
Приложение HammerDB

- реализует подмножество полной спецификации TPC-C, измененное для упрощения и облегчения выполнения рабочей нагрузки
- результаты теста тип-C HammerDB не сравнимы с результатами других тестов, использующих метрику tpmC
- результаты сравнимы, если они проведены в HammerDB
- отличие от TPC-C в том, что по умолчанию HammerDB работает без задержек на ввод данных и обдумывание. Тест тип-C запускает нагрузку TPC-C без задержек
- результат теста HammerDB тип-C: TPM и NOPM



Параметры для теста тип-С HammerDB

- официальный тест TPC-C имеет фиксированное количество пользователей на хранилище и использует время ввода и обдумывания (think time), чтобы рабочая нагрузка, создаваемая каждым пользователем, не была интенсивной
- HammerDB по умолчанию не использует время на ввода и обдумывание, и поэтому число Virtual Users примерно равно числу ядер на хосте с СУБД
- 4-5 складов на одного Virtual User будет минимальным значением для обеспечения равномерного распределения Virtual Users по складу



Утилита Go-TPC

- утилита написана на языке go, работает в командной строке, что позволяет автоматизировать ее запуск
- реализует тесты TPC-C, TPC-H, CHmark
- CHmark объединяет оба теста, использует схему таблиц TPC-C и упрощенную схему TPC-H
  - был создан для баз обслуживающих смешанную нагрузку: OLTP и одновременно OLAP
  - Для PostgreSQL рекомендуется переносить OLAP нагрузку на физические реплики

```
wget https://raw.githubusercontent.com/pingcap/go-tpc/master/install.sh
chmod +x install.sh
./install.sh
cd .g-tpc/bin
./go-tpc tpcc prepare --warehouses 1 -d postgres -U postgres -p postgres -H 127.0.0.1 -P 5432
./go-tpc tpcc run --time 30s -d postgres -U postgres -p postgres -H 127.0.0.1 -P 5432
```

Практика

- Часть 1. Стандартный тест pgbench
- Часть 2. Использование pgbench с собственным скриптом
- Часть 3. Использование утилиты sysbench
- Часть 4. Использование приложения HammerDB
- Часть 5. Использование приложения Go-TPC

tantor 2

2-1

Память

Оперативная память

Работа с памятью осуществляется страницами

- размер обычной страницы 4Кб задан аппаратно
- размер TLB (Translation Lookaside Buffer, буфер ассоциативной трансляции) 2-4Кб
- частота непопаданий в TLB 0.01-1%
- при попадании в TLB скорость доступа к памяти 1/2-1 такт процессора
- если ссылка отсутствует в TLB, используется медленный механизм преобразования
- непопадания обрабатываются за 10-100 тактов процессора
- размер обычной страницы памяти выдает команда:

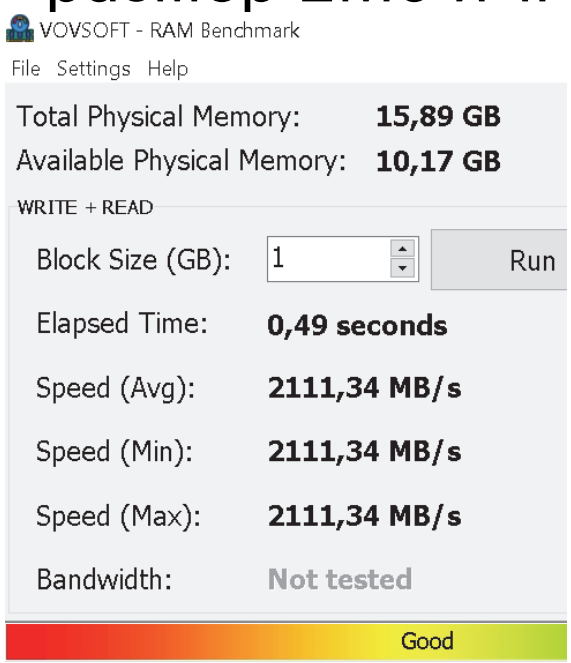
```
astra@tantor:~$ getconf PAGE_SIZE
4096
```

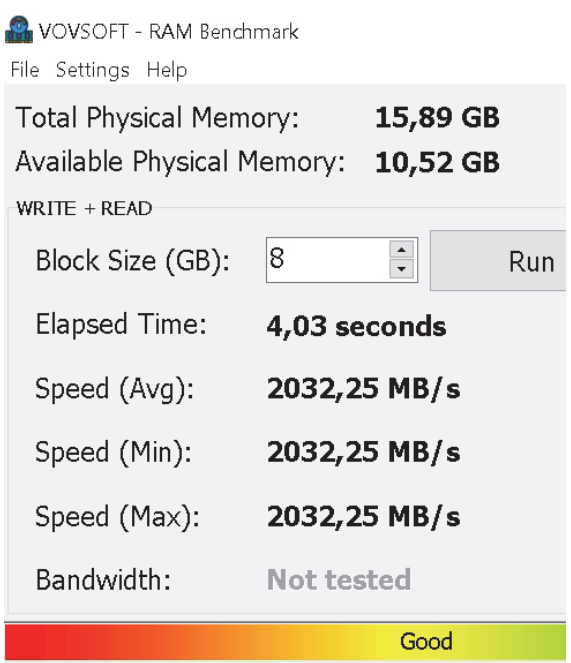
Виртуальная адресация памяти

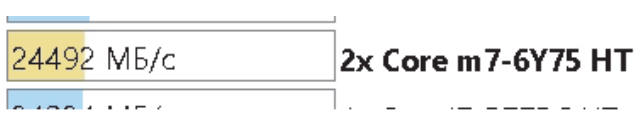
- операционная система и процессы работают с виртуальным адресным пространством
- MMU (memory management unit, блок управления памятью) делит виртуальное адресное пространство на участки одинакового размера, называемые страницами
- преобразование виртуального адреса в физический должно быть максимально быстрым
- операционная система должна прозрачно сохранять (вытеснять) содержимое физической памяти на внешний носитель и читать обратно (swapping)

Размер страниц памяти

- процессоры поддерживают 2 типа страниц, обычные и огромные (Huge Pages)
- На архитектуре x86-64 огромные страницы могут иметь размер 2Мб и 1Гб





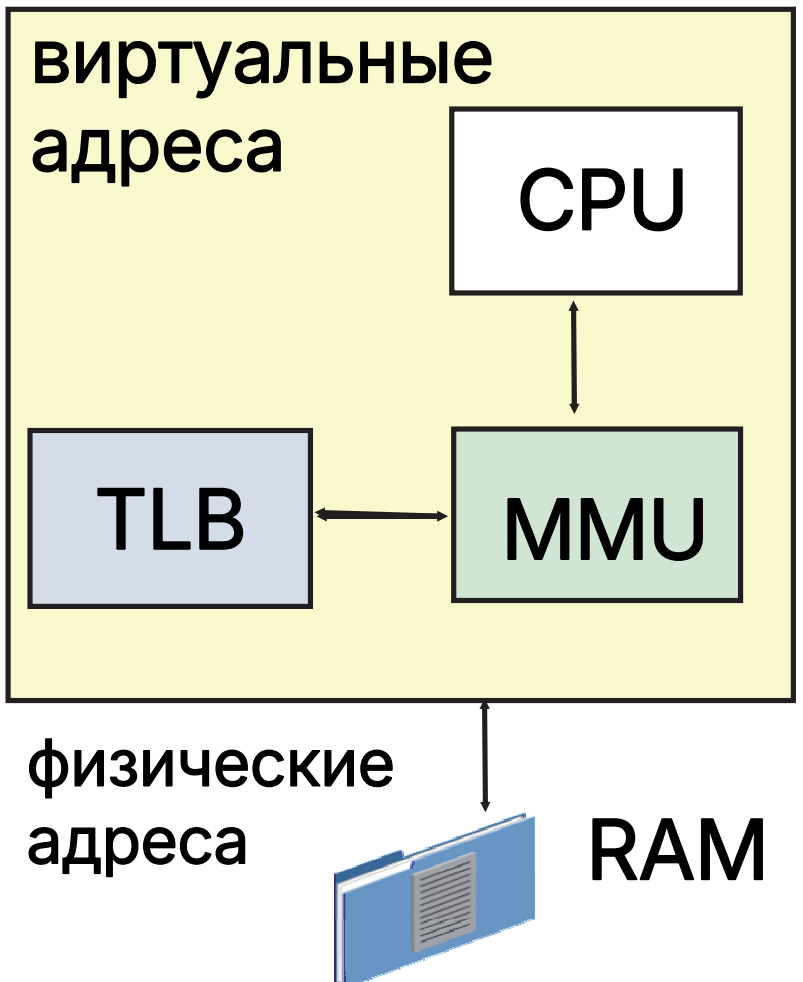


Размер буфера ассоциативной трансляции (TLB)

- организация TLB зависит от процессора (поколения, архитектуры)
- производительность при выборе 2M или 1G страниц на разных процессорах может существенно различаться
- У процессоров Intel Sky Lake, Cofee Lake, Cascade Lake в L2 TLB хранится 1536 ссылок на страницы размером 4K и 2M и 16 ссылок на страницы 1G
- у процессоров семейства Intel Sunny Cove количество ссылок 2048, из них может использоваться до 1024 ссылок на страницы размером 2M или 1G

Огромные страницы (Huge Pages)

- количество Huge Pages нужно установить вручную в параметре операционной системы `vm.nr_hugepages`
- устанавливать можно и нужно с небольшим запасом
- изменять количество Huge Pages можно без перезагрузки



Использование Huge Pages

Статистика использования:

- `cat /proc/meminfo | grep Huge`
  - `HugePages_Rsvd`: используется
  - `HugePages_Free`: можно использовать
  - `HugePages_Total`: доступно

```
root@tantor:~# cat /proc/meminfo | grep Huge
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
FileHugePages:         0 kB
HugePages_Total:     300
HugePages_Free:      281
HugePages_Rsvd:       72
HugePages_Surp:        0
Hugepagesize:       2048 kB
Hugetlb:          614400 kB
```

Прозрачные огромные страницы (Transparent Huge Pages)

- замедляют работу процессов СУБД
- не выделены, если
  - `AnonHugePages:  0 kB`
- проверить включено ли использование:

```
root@tantor:~# cat
/sys/kernel/mm/transparent_hugepage/enabled
always [madvise] never
root@tantor:~# cat
/sys/kernel/mm/transparent_hugepage/defrag
always defer defer+madvise [madvise] never
```

Использование экземпляром Huge Pages

- Huge Pages используются для разделяемого пула буферов и памяти Dynamic Shared Memory (DSA), которую используют параллельные процессы
- память выделяется способом задаваемым параметром `shared_memory_type`.
- в linux выделение памяти большими страницами поддерживается только способом `mmap` ("анонимные" страницы).
- Huge Pages не вытесняются в своп.
- если этого объема памяти будет недостаточно, то параллельные процессы будут дополнительно выделять (а потом освобождать) память страницами 4Кб и способом, задаваемым в параметре `dynamic_shared_memory_type`:

```
postgres=# select name, setting, enumvals from pg_settings where name like '%memory_type';
            name             | setting |     enumvals
-----------------------------+---------+------------------
 dynamic_shared_memory_type | posix   | {posix,sysv,mmap}
 shared_memory_type         | mmap    | {sysv,mmap}
```

Размер Резидентного Набора (RSS)

- RSS (резидентный размер)- объем памяти, выделенных процессу и в находящихся в физической памяти
- PSS (пропорциональный размер) дает полноценное представление о распределении физической памяти между процессами и разделяемыми библиотеками
- привлекательность процесса для `oom kill`:
  - `/proc/'$PID'/oom_score`

2-2

Нехватка памяти

Out Of Memory (OOM)

- процессам посылается сигнал `SIGKILL`
- лучшим кандидатом считается процесс, который
  - освободит максимум памяти
  - является наименее важным для системы
  - в идеале должен остановиться один процесс
- оценка для каждого процесса заранее расчитывается и записана в `/proc/PID/oom_score`
- сообщения об остановке процессов записываются в журнал операционной системы

Параметр `oom_score_adj`

- значение устанавливается после запуска для каждого процесса:

```
› echo -900 >  /proc/58253/oom_score_adj
```

- изменяет `/proc/PID/oom_score` для этого процесса
- значение -1000 для процесса `postgres` устанавливается в файле службы СУБД Tantor `tantor-se-server-16.service`
- у остальных процессов экземпляра по умолчанию не меняется и установлено в ноль

Параметр vm.overcommit_memory

- имеет три значения:
  - 0 - значение по умолчанию. Выделяется столько памяти, сколько запросит процесс.
  - 1 - с СУБД не используется.
  - 2 - отказ в выделении памяти, если суммарный объем выделенной памяти превысит размер пространства подкачки плюс объем физической памяти умноженной на значение в процентах параметра `vm.overcommit_ratio` (по умолчанию 50%) или абсолютное значение заданное параметром `vm.overcommit_kbytes`
- **если раздел подкачки отключён и `vm.overcommit_ratio<100`, то устанавливать значение 2 не стоит, оно должно быть 0**
- значение параметра `vm.overcommit_ratio` не играет роли при значениях параметра `vm.overcommit_memory` равных 0 или 1

Установка значений overcommit и swap

- при обслуживании в linux только СУБД стоит установить значения **`vm.overcommit_ratio=100`**, **`vm.overcommit_memory=2`**
  - при таких значениях срабатывание oom kill маловероятно и будет использоваться вся физическая память
  - станет возможным отключить раздел подкачки без увеличения вероятности срабатывания oom kill
  - если установить значение `vm.overcommit_ratio>100` увеличивается вероятность срабатывания oom kill
- общий размер виртуальной памяти, который может быть выделен определяется по формуле:
  `CommitLimit =(total_RAM-total_huge_TLB)* vm.overcommit_ratio/100+total_swap`
- активное использование раздела подкачки занимает пропускную способность шины ввода-вывода

Параметр `vm.swappiness`

- по умолчанию значение равно 60
- изменить значение можно без рестарта операционной системы
- оптимальное значение значение около 10
- **значение 0 не стоит использовать**
- влияет на то какие части памяти будут кандидатами на вытеснение в swap:
  - `anon_prio = swappiness;`
  - `file_prio = 200 - anon_prio;`
- Пример сообщения в журнале linux об остановке процесса:

```
Out of memory: Kill process 58302 (postmaster) score 837 or sacrifice child
Killed process 58302 (postmaster) total-vm:72328760kB, anon-rss:55470760kB, file-rss:4753180kB
```

Дедупликация страниц памяти (KSM)

Kernel Same-page Merging (дедупликация страниц памяти):

- может объединять страницы локальной памяти anon-rss разных процессов
- не объединяет file-rss и разделяемую память
- используется в виртуализации, а не с СУБД
- память сканирует в поисках одинаковых страниц служба linux

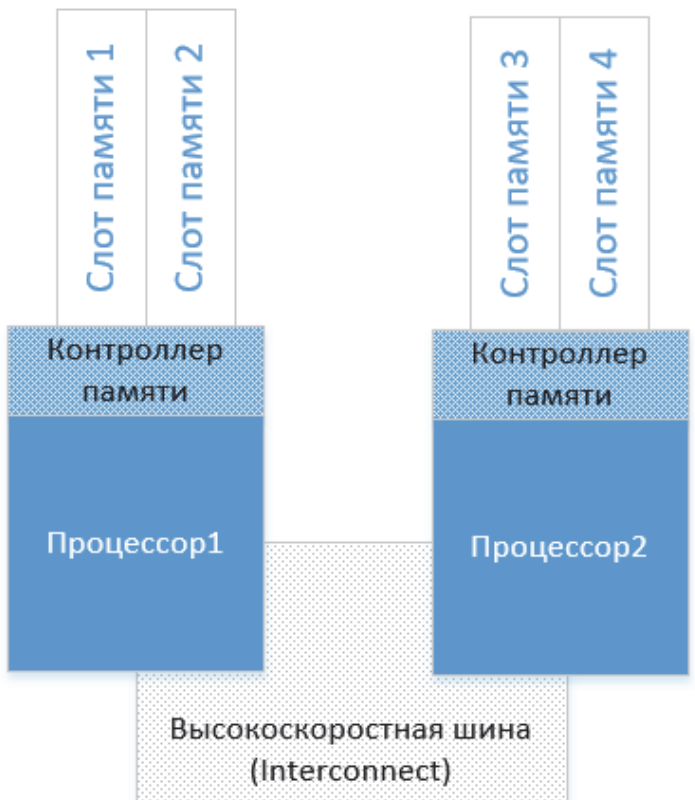
Проверить, что KSM отключён можно командой:

```
cat /sys/kernel/mm/ksm/run
0
```

Неравномерный доступ к памяти (NUMA)

Non-Uniform Memory Access (неравномерный доступ к памяти):

- каждый процессор имеет локальную физическую память и получает к ней доступ обычным образом через свой контроллер памяти. Помимо этого каждый процессор имеет доступ к локальной памяти других процессоров через более медленную шину ввода-вывода
- PostgreSQL оптимизирован для работы с однородным доступом к физической памяти и не оптимизирован для работы с NUMA



Параметр `enable_large_allocations`

- параметр СУБД Tantor который увеличивает размер StringBuffer с 1 гигабайта до 2 гигабайт

```
postgres=# select * from pg_settings where name like '%large%'\gx
name        | enable_large_allocations
setting     | off
category    | Resource Usage / Memory
short_desc | whether to use large memory buffer greater than 1Gb, up to 2Gb
context     | superuser
vartype     | bool
boot_val    | off
```

- может устанавливаться на уровне сессии и утилитами pg_dump, pg_dumpall

```
postgres@tantor:~$ pg_dump --help | grep alloc
  --enable-large-allocations   enable memory allocations with size up to 2Gb
```

- проблема возникает со строкой таблицы config приложений 1С:ERP, Комплексная автоматизация, Управление производственным предприятием

Выделение локальной памяти процессами экземпляра

- в PostgreSQL реализована логика управления памяти через "контексты памяти"
- используется вызов palloc(), который выделяет память в области памяти, называемой MemoryContext, который ещё называют "Arena"
- контексты памяти образуют иерархию
- TopMemoryContext - корень иерархии контекстов памяти серверного процесса
- **при запросе на выделение памяти, выделяется размер памяти являющийся ближайшей степенью 2 в большую сторону от запрошенного размера**
- если в контексте недостаточно свободной памяти, то к нему добавляется память **в двойном размере от начального**

ОШИБКА: invalid memory alloc request size

- причина ошибки: попытка выделить блок памяти, превышающий ограничение 2Гб или 1Гб-1, установленный макросом `MaxAllocSize`
- в текстовых функциях типа lpad, rereat вставлены проверки, которые выдают ошибку: requested length too large
- если размер памяти после текста ошибки больше 2Гб, это означает, что размер памяти, который должен быть выделен рассчитан неверно и может указывать на наличие повреждения записей в блоках данных
  - также расчет может быть неверно выполнен из-за ошибок в библиотеках расширений, загруженных в память процесса
- пример команд, приводящих к ошибкам:

```
postgres=# create table a as select repeat('a', 1024 * 1024 * 1024 - 5);
ERROR:  invalid memory alloc request size 1073741887
postgres=# select repeat('x', 1024 * 1024 * 1024);
ERROR:  requested length too large
```

2-3

Страничный кэш

Страничный кэш linux

Страничный кэш (кэш операционной системы, файловый кэш):

- кэш 4Кбайтных страниц отображаемых на файлы в файловой системе
- Под страничный кэш linux может использовать всю свободную (не занятую процессами и ядром) физическую память, за исключением части, размер которой косвенно определяется параметром `vm.min_free_kbytes`
  - параметр не позволяет использовать под страничный кэш всю физическую память
  - значение по умолчанию не больше 66Мб

Доля изменённых ("грязных") страниц в кэше

- процент изменённых ("грязных") страниц в физической памяти задаётся параметрами `vm.dirty_background_ratio` и `vm.dirty_ratio`
- процент берётся от объема незанятой процессами и ядром физической памяти (available), а не от объема всей физической памяти.
- также есть параметры `vm.dirty_background_bytes` и `vm.dirty_bytes`, задающие абсолютные значения, по умолчанию ноль
- текущие значения порога срабатывания начала записи грязных страниц на их исходные места (сначала в фоновом режиме без блокировки, потом с блокировкой), в количестве страниц:

```
root@tantor:~# cat /proc/vmstat | grep dirty
nr_dirty 4
nr_dirty_threshold 152308
nr_dirty_background_threshold 76061
```

Фрагментация памяти

- на фрагментацию виртуального адресного пространства памяти указывает то, что кусков памяти размером больше 16Кб выше 16Мб нет:

```
root@tantor:~# cat /proc/buddyinfo
Node 0,zone      DMA      1      0      0      1      2      1      1      0      1      1      3
Node 0,zone    DMA32   3173    856    529      0      0      0      0      0      0      0      0
Node 0,zone   Normal  19030   8688   7823      0      0      0      0      0      0      0      0
```

- пример, когда большая часть памяти кусками размером не меньше 4Мб:

```
root@tantor:~# cat /proc/buddyinfo
Node 0,zone      DMA      0      0      0      0      0      0      0      0      0      1      3
Node 0,zone    DMA32      1      2      4      2      3      4      2      2      2      2    863
Node 0,zone   Normal      1      0      3     91     14     10      9      5      5     23    352
```

- каждая зона делится на части адресного пространства памяти размером (4096 байт *2^n) : 4Кб, 8Кб, 16Кб, 32Кб, 64Кб, 128Кб, 256Кб, 512Кб, 1Мб, 2Мб, 4Мб.

Дефрагментация памяти

- значение по умолчанию для минимального размера свободной оперативной памяти может быть недостаточным:

```
root@tantor:~# sysctl -a | grep min_free
vm.min_free_kbytes = 67584
```

- дефрагментация запускается при уменьшении свободной памяти ниже `vm.min_free_kbytes`
- рекомендуется установить в 2% от размера физической памяти
- параметр `vm.watermark_scale_factor` задает вторую границу для процесса дефрагментации
- значение по умолчанию **0.1% (число 10)** от размера **свободной** физической памяти:

```
root@tantor:~# sysctl -a | grep watermark
vm.watermark_boost_factor = 15000
vm.watermark_scale_factor = 10
```

- рекомендуется установить 1% (число 100)

Длительность удержания грязных страниц

- `vm.dirty_expire_centisecs` - сколько буфер может быть грязным, прежде чем будет помечен для записи
- `vm.dirty_writeback_centisecs` - период ожидания между записями на диск
- `net.ipv4.tcp_timestamps` может уменьшить периодические задержки из-за генерации временных меток

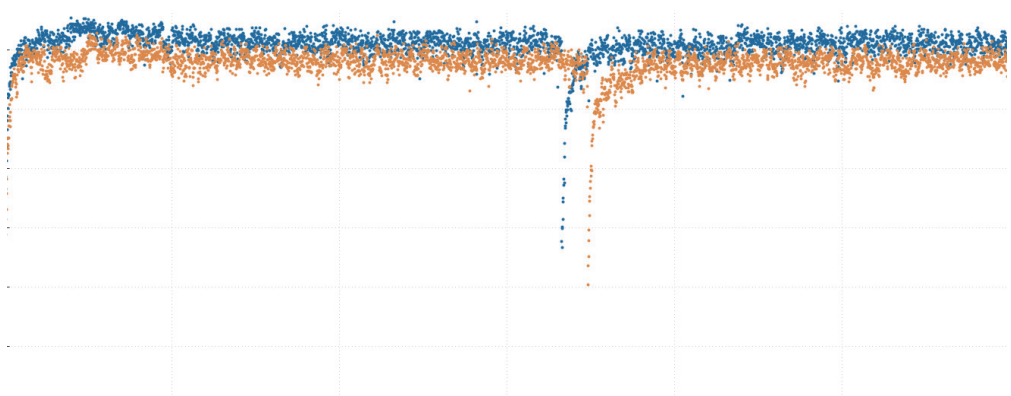
Пример настроек:

```
vm.dirty_expire_centisecs = 500
vm.dirty_writeback_centisecs = 250
vm.swappiness = 10
vm.dirty_ratio = 10
vm.dirty_background_ratio = 3
net.ipv4.tcp_timestamps=0
```

Параметр backend_flush_after

- количество грязных блоков, вытесняемых из буферного кэша каждым серверным процессом, по достижении которого будет послана команда на вытеснение страниц файлов, которые соответствуют этим блокам из страничного кэша
- ограничивает объем грязных страниц в страничном кэше linux и уменьшает вероятность проседания производительности при выполнении вызовов fsync по файлам данных в конце контрольной точки

```
\dconfig *flush*
List of configuration parameters
       Parameter         | Value
-------------------------+-------
 backend_flush_after     | 0
 bgwriter_flush_after    | 512kB
 checkpoint_flush_after  | 256kB
 wal_writer_flush_after  | 1MB
```



Практика

- Часть 1. Запуск экземпляра с огромными страницами
- Часть 2. Изменение значения oom_score
- Часть 3. Выгрузка длинных строк утилитой `pg_dump`
- Часть 4. Нехватка памяти
- Часть 5. Включение подкачки (swap)
- Часть 6. Страничный кэш

3-1

Процессоры

Simultaneous Multi-Threading (SMT) и Hyper-Threading (HT)

- Hyper-Threading - название технологии Intel
- Simultaneous Multi-Threading - название технологии AMD
- одно физическое ядро процессора определяется linux как два виртуальных (логических) ядра
- реализует концепцию одновременной многопоточности
- проверка сколько имеется потоков на ядро и имеется ли поддержка SMT и активен ли SMT:

```
root@tantor:~# lscpu | grep Thread
Thread(s) per core: 1
root@tantor:~# cat /sys/devices/system/cpu/smt/active
0
root@tantor:~# cat /sys/devices/system/cpu/smt/control
notsupported
```

Привязка процесса к процессору (CPU affinity)

- увеличивает вероятность попадания данных процессов в кэши процессоров (TLB и другие)
- можно не рассматривать, если:
  - процессор один
  - нагрузка на ядра процессоров невысокая (меньше 80%)
- пример привязки работающего процесса:

```
root@tantor:~# ps -ef | grep check
postgres  2181  2179  0       00:00:01 postgres: checkpointer
root@tantor:~# taskset -p 1 2181
pid 2181's current affinity mask: f
pid 2181's new affinity mask: 1
```

Просмотр списка процессов утилитой ps

- утилита командной строки ps выдаёт список работающих процессов
- Используя пайпы можно форматировать вывод утилиты
- Пример первых 4 процессов в выводе, включая первую строку заголовка, описывающего значение в столбцах:

```
ps -A -o pid,psr,cmd | head -4
    PID PSR CMD
      1   2 /sbin/init splash
      2   1 [kthreadd]
      3   0 [pool_workqueue_release]
```

- пример вывода процессов с именем postgres и их сортировки по столбцу pss в убывающем порядке:

```
ps -C postgres -o pcpu,vsz,rss,pss,rops,wops,cmd --sort -pss | head -4
%CPU    VSZ   RSS   PSS  ROPS  WOPS CMD
 0.0 231628 18416 14263     5  3069 postgres: checkpointer
 0.0 233712 16172  9491    87    26 postgres: postgres postgres [local] idle
 0.0 231516  6768  4301    34    98 postgres: walwriter
```

Запись и просмотр метрик утилитой `atop`

- для мониторинга процессов и текущей нагрузки удобно использовать утилиту командной строки `top`
- преимущество утилиты `top` в том, что она обычно установлена по умолчанию
- утилита atop позволяет записать в бинарный файл показатели работы операционной системы и, затем, визуализировать собранные показатели:

```
root@tantor:~# apt install atop -y
root@tantor:~# dpkg -S /usr/bin/atop
atop: /usr/bin/atop
root@tantor:~# atop -w /atop.record 1 15
root@tantor:~# atop -r /atop.record
```

```
ATOP - tantor                    2024/12/08  15:00:32                    ----------                    1s elapsed
PRC | sys    0.07s | user   0.44s | #proc    167 | #tslpi   228 | #tslpu    57 | #zombie    0 | #exit      0 |
CPU | sys      6% | user     52% | irq       0% | idle    142% | wait      0% | guest     0% | curf 1.51GHz |
cpu | sys      4% | user     25% | irq       0% | idle     71% | cpu000 w  0% | guest     0% | curf 1.51GHz |
cpu | sys      2% | user     27% | irq       0% | idle     71% | cpu001 w  0% | guest     0% | curf 1.51GHz |
CPL | numcpu     2 | avg1    0.91 | avg5    0.77 | avg15   0.48 |              | csw    2863 | intr    1279 |
MEM | tot     1.9G | free  582.8M | cache 770.1M | dirty   6.3M | buff   66.1M | slab  104.2M | pgtab  11.3M |
MEM | numnode    1 |              | shmem  43.5M | shrss   0.0M | shswp   0.0M | tcpsk   0.0M | udpsk   0.0M |
SWP | tot     0.0M | free    0.0M | swcac   0.0M |              |              | vmcom   1.8G | vmlim 989.5M |
PAG | numamig    0 | migrate    0 | pgin       0 | pgout    239 | swin       0 | swout      0 | oomkill    0 |
PSI | cpusome   7% | memsome   0% | memfull    0% | iosome     0% | iofull     0% | cs      6/3/5 | ms      0/0/0 |
DSK |          sda | busy      8% | read       0 | write     69 | MBr/s    0.0 | MBw/s    0.9 | avio 1.04 ms |

    PID SYSCPU USRCPU  RDELAY BDELAY  VGROW  RGROW  RDDSK  WRDSK  ST EXC  THR S  CPUNR  CPU CMD          1/2
   1270  0.00s  0.21s  0.00s  0.00s     0B 512.0K     0B     0B  --   -   14 S      1  24% syslog-ng
    244  0.01s  0.13s  0.05s  0.00s   8.0M   2.1M     0B 828.0K  --   -    1 R      0  16% systemd-journa
   4343  0.02s  0.07s  0.01s  0.00s     0B     0B     0B  12.0K  --   -    1 R      1  10% atop
    319  0.02s  0.02s  0.03s  0.00s     0B     0B     0B 364.0K  --   -    2 R      0   5% auditd
   2463  0.00s  0.01s  0.04s  0.00s     0B     0B     0B     0B  --   -   10 R      1   1% fly-start-menu
     23  0.01s  0.00s  0.00s  0.00s     0B     0B     0B     0B  --   -    1 S      1   1% ksoftirqd/1
     30  0.01s  0.00s  0.01s  0.00s     0B     0B     0B     0B  --   -    1 R      0   1% kauditd
   2448  0.00s  0.00s  0.00s  0.00s     0B     0B     0B     0B  --   -    6 S      0   0% org_kde_powerd
   2493  0.00s  0.00s  0.00s  0.00s     0B     0B     0B     0B  --   -    4 S      1   0% fly-reflex-ser
   2450  0.00s  0.00s  0.00s  0.00s     0B     0B     0B     0B  --   -    3 S      1   0% baloo_file
    495  0.00s  0.00s  0.00s  0.00s     0B     0B     0B     0B  --   -    3 S      0   0% NetworkManager
   2388  0.00s  0.00s  0.02s  0.00s     0B     0B     0B     0B  --   -    4 S      0   0% VBoxClient
     15  0.00s  0.00s  0.00s  0.00s     0B     0B     0B     0B  --   -    1 S      0   0% ksoftirqd/0
```

Переключения контекста выполнения

- добровольное переключение контекста:
  - процесс не может выполнять свой код, так как ждет выполнения операции ввода-вывода, получения блокировки
- недобровольное переключение контекста:
  - процесс превышает время (timeslice) выделенное ему планировщиком
  - в соответствии с политикой, установленной для процесса планировщик имеет право приостановить выполнение процесса (вытеснить процесс с ядра процессора)
- постоянное большое число недобровольных переключений контекста указывает на слишком большую степень параллелизма, не соответствующую числу ядер процессоров

```
root@tantor:~# grep ctxt /proc/212233/status
voluntary_ctxt_switches:        0
nonvoluntary_ctxt_switches:     62728
```

Планировщик операционной системы

- По умолчанию CFS используется политика `SCHED_OTHER` с алгоритмом:
  - CFS - до версии ядра 6.6 linux
  - EEVDF - заменяет CFS начиная с версии 6.6 ядра linux, устраняет задержку в начале выполнения задачи
- минимальное время работы процесса до вытеснения: не ниже `kernel.sched_rr_timeslice_ms`, по умолчанию 1/10 секунды, значение можно менять
- политики могут принимать во внимание приоритет процесса или nice или игнорировать их
- политики играют роль когда процессора полностью нагружены
- политику можно менять для каждого процесса:

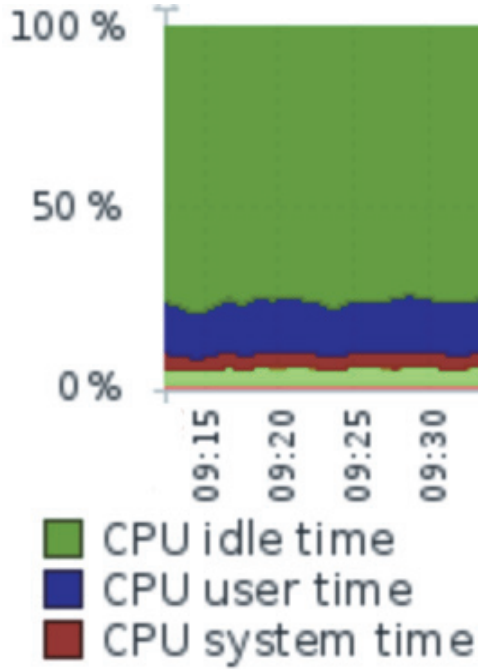
```
root@tantor:~# chrt -r -p 10 96878
root@tantor:~# chrt -p 96878
pid 96878's current scheduling policy: SCHED_RR
pid 96878's current scheduling priority: 10
```

Фоновые рабочие процессы

- `max_worker_processes` (по умолчанию 8) максимальное число фоновых процессов на экземпляре
  - измерение параметра требует перезапуска экземпляра
  - на физических репликах значение параметра должно быть не меньше, чем на мастере.
- `max_parallel_workers` (по умолчанию 8) максимальное число фоновых процессов для обслуживания команд
- `max_parallel_workers_per_gather` (по умолчанию 2) ограничение степени параллелизма
- `parallel_leader_participation` (по умолчанию `on`) серверный процесс будет помогать фоновым процессам

Использование процессорного времени кодом приложений и ядра (пропорция USER/SYS)

- отношение User CPU time/System CPU time должно быть около 60/40
- если User больше это означает неэффективный код приложения
- если System больше - linux испытывает проблемы




```
postgres@tantor:~$ rm -rf $HOME/backup/1
postgres@tantor:~$ pg_basebackup -D $HOME/backup/1 -T $PGDATA/../u01=$HOME/backup/1/u01 -P
1622874/4472046 kB (36%), 1/2 tablespaces
```

Файл Правка Настройка Справка

```
top - 01:27:13 up 5 days, 14:52,  3 users,  load average: 1.39, 1.13, 0.87
Tasks: 174 total,   2 running, 172 sleeping,   0 stopped,   0 zombie
%Cpu(s): 48.6 us, 15.7 sy,  0.0 ni,  7.2 id, 26.8 wa,  0.0 hi,  1.6 si,  0.0 st
MiB Mem :   3913.6 total,    107.8 free,    532.1 used,   3273.7 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   2929.9 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
26812 postgres  20   0   20608   7648   6500 R  91.0   0.2   0:05.41 pg_basebackup
26813 postgres  20   0  218408  16860  14328 D  31.6   0.4   0:02.65 postgres
26808 root      20   0       0      0      0 D   4.7   0.0   0:00.63 kworker/u4:1+flush-8:0
   47 root      20   0       0      0      0 S   4.3   0.0   0:58.18 kswapd0
```

```
postgres@tantor:~$ pg_basebackup -D $HOME/backup/1 -T $PGDATA/../u01=$HOME/backup/1/u01 -P -r 500k
 41729/4472046 kB (0%), 0/2 tablespaces
```

Файл Правка Настройка Справка

```
top - 21:59:34 up 6 days, 11:24,  3 users,  load average: 0.83, 0.36, 0.14
Tasks: 173 total,   2 running, 171 sleeping,   0 stopped,   0 zombie
%Cpu0  : 92.3 us,  7.7 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  1.0 us,  0.0 sy,  0.0 ni, 99.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3913.6 total,    782.9 free,    540.0 used,   2590.7 buff/cache
MiB Swap:      0.0 total,      0.0 free,      0.0 used.   2918.3 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
31280 postgres  20   0   20608   7728   6580 R 100.0   0.2   1:17.32 pg_basebackup
  450 root      20   0 1578688  12344      0 S   0.3   0.3  10:14.54 containerd
```

Источник времени (clock source)

```
linux/arch/x86/kernel/early-quirks.c
/*
 * HPET on the current version of the Baytrail platform has accuracy
 * problems: it will halt in deep idle state - so we disable it.
 *
 * More details can be found in section 18.10.1.3 of the datasheet:
 http://www.intel.com/content/dam/www/public/us/en/documents/datasheets/atom-z8000-datasheet-vol-1.pdf
 */
{ PCI_VENDOR_ID_INTEL, 0x0f00,
  PCI_CLASS_BRIDGE_HOST, PCI_ANY_ID, 0, force_disable_hpet},
```

- используются и ядром linux и приложениями для получения меток времени
- во время загрузки linux проверяет доступные источники времени и выбирает один для использования
- наиболее быстрый источник: Time Stamp Counter (TSC)
- ACPI Power Management Timer (ACPI_PM) медленнее в несколько раз
- доступные источники:

```
cat /sys/devices/system/clocksource/clocksource0/available_clocksource
tsc hpet acpi_pm
```

- используемый источник:

```
cat /sys/devices/system/clocksource/clocksource0/current_clocksource
acpi_pm
```

- сообщения при выборе источника времени можно посмотреть в сообщениях ядра командой: **dmesg | grep tsc**

Сравнение источников времени

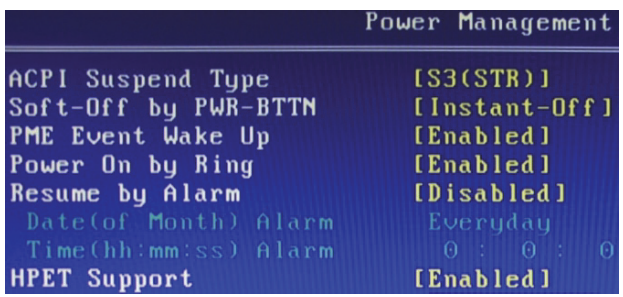
- для проверки скорости источника времени можно использовать программу, запрашивающую время много раз

для acpi_pm:

```
time ./clock_timing
real    0m38.889s
user    0m15.760s
sys     0m23.126s
```

Для tsc:

```
time ./clock_timing
real    0m13,967s
user    0m13,938s
sys     0m0,008s
```



- **разница в скорости между acpi_pm и tsc существена: 2,8 раза**
- `real`: время от вызова программы до завершения. real включает в себя user и sys и может быть больше их суммы, если программа вытеснялась планировщиком (involuntary context switches).
- `user`: время выполнения кода программы.
- `sys`: время выполнения кода ядра linux (работа с оборудованием, памятью, файлами, потоками, сетью)

Сравнение источников времени в PostgreSQL

- источник времени активно используется при включении параметров конфигурации `track_wal_io_timing`, `track_io_timing`, `track_commit_timestamp`
- пример влияния выбора источника времени на время выполнения команды `explain analyze`:

```
postgres=# \! sudo sh -c 'echo acpi_pm > /sys/devices/system/clocksource/clocksource0/current_clocksource'
postgres=# explain analyze select count(pk) from t;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=1791.00..1791.01 rows=1 width=8) (actual time=836.212..836.228 rows=1 loops=1)
   ->  Seq Scan on t  (cost=0.00..1541.00 rows=100000 width=8) (actual time=0.019..408.416 rows=100000 loops=>
 Planning Time: 0.056 ms
 Execution Time: 836.334 ms
(4 rows)
postgres=# \! sudo sh -c 'echo tsc > /sys/devices/system/clocksource/clocksource0/current_clocksource'
postgres=# explain analyze select count(pk) from t;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Aggregate  (cost=1791.00..1791.01 rows=1 width=8) (actual time=308.180..308.187 rows=1 loops=1)
   ->  Seq Scan on t  (cost=0.00..1541.00 rows=100000 width=8) (actual time=0.022..153.991 rows=100000 loops=>
 Planning Time: 0.375 ms
 Execution Time: 308.373 ms
(4 rows)
```

Замена источника времени

- при загрузке linux может выбираться источник времени acpi_pm, который медленнее tsc
- для изменения источника на tsc нужно добавить в файл `/etc/default/grub` после `quiet splash` параметры:

```
clocksource=tsc nohpet processor.max_cstate=1 intel_idle.max_cstate=0
```

- и выполнить команду `update-grub`
- после перезагрузки проверить что используется `tsc` и `max_cstate=0`:

```
cat /sys/devices/system/clocksource/clocksource0/current_clocksource
tsc
cat /sys/module/intel_idle/parameters/max_cstate
0
```

- проверить скорость и стабильность источника времени можно утилитой командной строки `pg_test_timing`
  - утилита кроме выдаёт распределение задержек в скорости отдачи времени

3-2

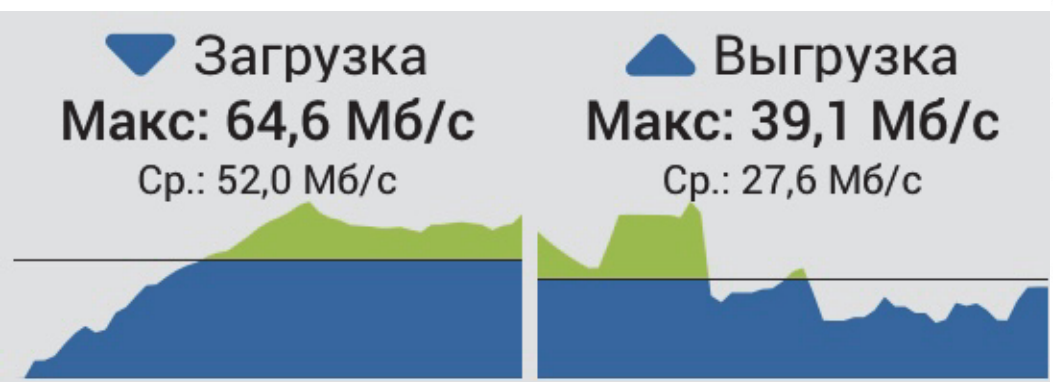
Сеть

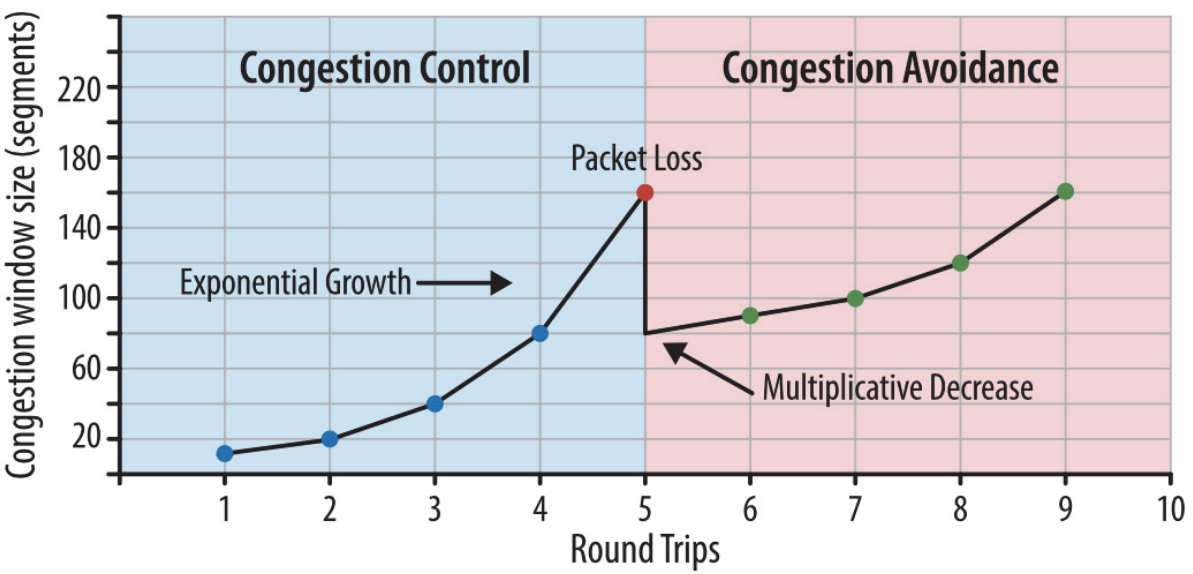
Основные параметры сети

- основные параметры:
  - пропускная способность
  - сетевая задержка
  - наличие и частота сетевых сбоев
- пропускная способность обычно не является узким местом
- экземпляр использует:
  - unix sockets для локальных соединений
  - TCP/IP для соединений через сетевые интерфейсы
- сетевая задержка важна при синхронной фиксации транзакций с подтверждением репликой

Алгоритмы congestion и slow start

- `net.ipv4.tcp_available_congestion_control` определяет алгоритм выбора скорости передачи данных отдельно по каждому сетевому соединению (сокету)
- `net.ipv4.tcp_slow_start_after_idle=0` отключает медленный старт
- без подтверждения передается 3 пакета для размера пакета 1095-2190 байт





Алгоритм BBR (Bottleneck Bandwidth и Round Trip Time)

- по умолчанию используется алгоритм CUBIC
- CUBIC чувствителен к потерям пакетов, BBR не чувствителен
- BBR при полной нагрузке на среду передачи потребляет всю доступную полосу пропускания и вытесняет другие сокеты, которые используют cubic и другие алгоритмы
- включение:
  - `sysctl -w net.ipv4.tcp_congestion_control=bbr`
  - `sysctl -w net.core.default_qdisc=fq`
  - `net.ipv4.tcp_timestamps = 1`

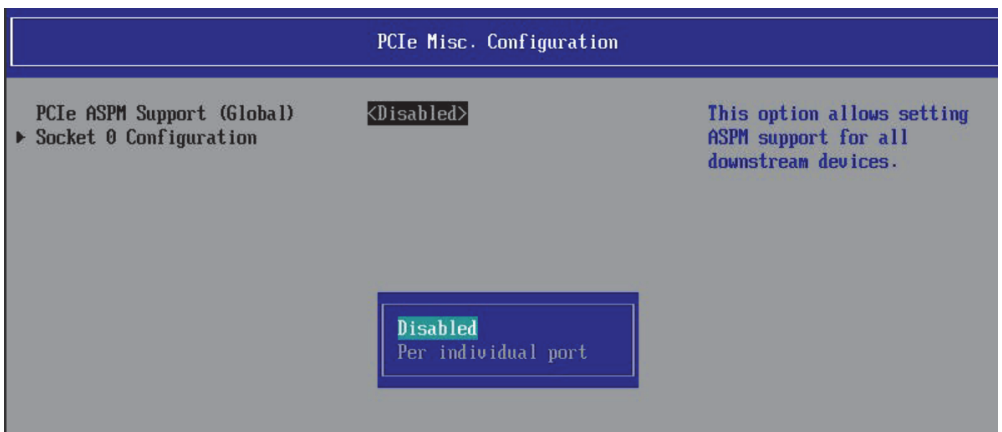
Параметры сетевых соединений

- `tcp_user_timeout` сколько переданные данные могут оставаться неподтверждёнными, прежде чем будет принято решение о принудительном закрытии TCP-соединения
- `client_connection_check_interval` (в миллисекундах) интервал между проверками в процессе выполнения команды путем опроса состояния сокета, по которому не передаются данные

```
root@tantor:~# sysctl -a | grep keepalive
net.ipv4.tcp_keepalive_intvl = 75 -> 20
net.ipv4.tcp_keepalive_probes = 9 -> 5
net.ipv4.tcp_keepalive_time = 7200 -> 240

net.ipv4.tcp_slow_start_after_idle = 1 -> 0
net.ipv4.tcp_retries2 = 15 -> 3
net.ipv4.tcp_timestamps = 1 -> 0
```

Параметры энергосбережения

- в операционной системе не нужно включать
- из-за ошибок в реализации может снижать производительность
- часть настроек в firmware




```
root@student:~# cat /sys/module/pcie_aspm/parameters/policy
[default] performance powersave powersupersave
root@student:~# echo performance > /sys/module/pcie_aspm/parameters/policy
root@student:~# cat /sys/module/pcie_aspm/parameters/policy
default [performance] powersave powersupersave
```

Практика

- Часть 1. Стандартный тест pgbench
- Часть 2. Привязка процессов к ядру процессора
- Часть 3. Переключения контекстов выполнения
- Часть 4. Мониторинг нагрузки на процессор
- Часть 5. Сбор статистик в файл и его просмотр утилитой `atop`
- Часть 6. Источник времени linux
- Часть 7. Сетевые соединения
- Часть 8. Замена политики планирования и проверка работы планировщика

4

Система хранения

Дисковая подсистема

- синонимы: ввод-вывод, система хранения
- наиболее нагруженный ресурс СУБД из основных ресурсов:
  - диск
  - процессор
  - память
  - сеть
- экземпляр может использовать прямой ввод/вывод, в 16 версии не обеспечивает лучшей отказоустойчивости и производительности
- настройка дисковой подсистемы включает:
  - выбор систем хранения (например, SSD или HDD)
  - выбор i/o scheduler
- настройка параметров кластера и табличных пространств
- создание точек монтирования, выбор файловых систем, менеджеров томов, параметров монтирования
- мониторинг ввода/вывода и смежные настройки

HDD, SSD, NVMe

- SSD (solid-state drive) - твердотельный (на микросхемах) "диск"
- NVMe, (Non-Volatile Memory express, энергонезависимая память) - интерфейс доступа к SSD, подключенным по шине PCIe
- SATA (Serial Advanced Technology Attachment) транспортный протокол, который определяет взаимодействие между контроллером и устройствами хранения
- SAS (Serial Attached SCSI) - набор команд SCSI по физическому интерфейсу, аналогичному SATA
- AHCI (Advanced Host Controller Interface) стандарт, описывающий операции с контроллерами SATA
  - на каждый порт одна очередь глубиной до 32 команд
  - поддерживается горячая замена устройств
- M.2 (NGFF, Next Generation Form Factor) - общее название форм-фактора и физического интерфейса для SSD, имеет 4 линии PCIe с общей скоростью 4 или 8 гигабайт в секунду
- Скорость SATA - 600 мегабайт в секунду

Блочные устройства

- вид специальных файлов в linux, обеспечивающих интерфейс доступа к устройству (или обычному файлу)
- чтение-запись идет блоками равного размера
- располагаются в директории `/dev` смонтированной на виртуальной файловой системе `devtmpfs`
- в `/dev` находятся файлы только тех устройств, которые в настоящий момент доступны (подключены)
- действия при обнаружении устройств задаются в файлах директории `/lib/udev/rules.d`
- список блочных устройств:

```
root@tantor:~# ls -l /dev | grep br
brw-rw----  1 root    disk       7,    0 дата время loop0
brw-rw----  1 root    disk       8,    0 дата время sda
brw-rw----  1 root    disk       8,    1 дата время sda1
brw-rw----+ 1 root    cdrom     11,    0 дата время sr0
```

Планировщик ввода/вывода (I/O Scheduler)

- при использовании высокоскоростных устройств NVMe нет смысла использовать какой-либо планировщик, то есть стоит установить планировщик **`none`**
- контроллеры устройств памяти NVMe на шинах с минимальной задержкой PCIe справляются с потоком параллельных запросов и при этом не нагружают центральные процессоры и кэши процессоров, в отличие от кода планировщиков
- цель планировщиков: выстроить запросы от большого количества процессов в очередь, передавая контроллеру устройства хранения запросы один за другим, чтобы не перегружать контроллер
- современные планировщики: **`none mq-deadline kyber bfq`**

Изменение I/O Scheduler

- тип планировщика можно установить отдельно для разных устройств и их типов SSD или HDD
- посмотреть планировщик для блочного устройства:

```
root@tantor:~# cat /sys/block/sda/queue/scheduler
none [mq-deadline]
```

- поменять планировщик без перезагрузки на другой:

```
root@tantor:~# echo none > /sys/block/sda/queue/scheduler
```

- для создания постоянного правила создать или отредактировать файл `/etc/udev/rules.d/70-schedulerset.rules`

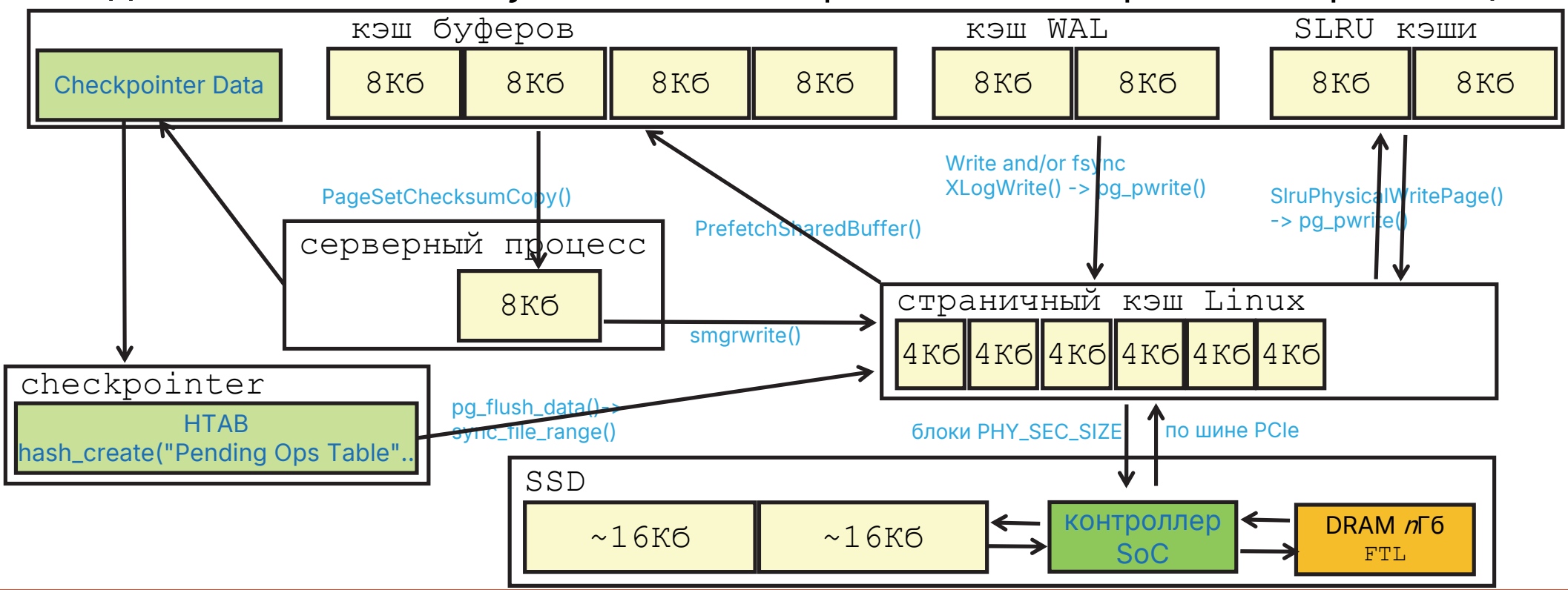
```
ACTION=="add|change", KERNEL=="sd[a-z]", TEST!="queue/rotational",
                                          ATTR{queue/scheduler}="none"
```

Физический сектор диска

- минимальная единица хранения, которую физическое устройство хранения данных может записывать атомарно
- обычно имеет размер 512Кбайт или 4Кбайт
- для NVMe, linux использует значение параметра Atomic Write Unit Power Fail (AWUPF), если он предоставляется оборудованием

Взаимодействие процессов экземпляра с диском

- блоки по 8Кб считываются в разделяемую память через страничный кэш (блоки по 4Кб)
- блоки записываются на диск через страничный кэш
- для записи используется оптимизированный алгоритм синхронизации



Синхронизация файлов данных с диском

- выполняет преимущественно checkpointer
- выполняется по диапазонам блоков системным вызовом: `sync_file_range(fd, offset, nbytes, SYNC_FILE_RANGE_WRITE)`
- если на системный вызов выдается ошибка записи, то экземпляр аварийно останавливается и блоки будут восстановлены по журналам WAL благодаря их полным образам (параметр конфигурации `full_page_writes=on`), если `data_sync_retry=off`

```
postgres=# \dconfig *flush*
List of configuration parameters
        Parameter         | Value
--------------------------+-------
 backend_flush_after      | 0
 bgwriter_flush_after     | 512kB
 checkpoint_flush_after   | 256kB
 wal_writer_flush_after   | 1MB
(4 rows)
```

```
postgres=# select backend_type name, sum(writes) buffers_written, round(sum(write_time)) w_time,
sum(writebacks) writebacks, sum(evictions) evictions, sum(fsyncs) fsyncs, round(sum(fsync_time))
fsync_time from pg_stat_io group by backend_type having sum(writes)> 0 or sum(writebacks)> 0 or
sum(fsyncs)>0 or sum(evictions)>0;
       name         | buffers_written | w_time | writebacks | evictions | fsyncs | fsync_time
--------------------+-----------------+--------+------------+-----------+--------+------------
 background worker |               7 |      0 |          0 |        43 |      0 |          0
 client backend    |         7451184 |  83639 |          0 |  15253670 |      0 |          0
 autovacuum worker |           61803 |    705 |          0 |     94784 |      0 |          0
 background writer |         2344704 |  26415 |    2344704 |           |      0 |          0
 checkpointer      |         5595043 |  99818 |    5598779 |           |  84627 |     843454
(5 rows)
```

Размер блока файловой системы

- должен быть равен размеру страницы - 4Кб
- файловая система ext4 лучший выбор
- параметры создаваемой по умолчанию ext4 оптимальны
- список параметров, с которыми смонтирована файловая система: **`cat /proc/fs/ext4/sda1/options`**

Параметр `wal_sync_method`

- системный вызов, который используется для того, чтобы гарантированно сохранить записи блоков в WAL файл, если значение параметра `fsync=on`
- `fdatasync` - значение по умолчанию для linux

```
postgres=# show fsync;
 fsync
-------
 on
postgres=# show wal_sync_method;
 wal_sync_method
-----------------
 fdatasync
```

Гарантия записи в WAL

- системные вызовы `fsync()` или `fdatasync()`, вызываемые после записи в WAL журнальной записи о COMMIT при `wal_sync_method = fdatasync` или `fsync` гарантируют, что запись блоков данных в WAL файл будет выполнена контроллером диска до возврата результата из этих функций
- перед записью в файл WAL он создается (переименовывается) полного размера и пока в него идёт запись его метаданные не меняются и гарантия записи в журнал файловой системы для записи в WAL не имеет значения

Быстрые фиксации изменений в журнале файловой системы ext4 (fast_commit)

- если WAL находятся на отдельной файловой системе, то быстрая фиксация не увеличит производительность
- параметр `commit` (по умолчанию 5 секунд) задает частоту вызова sync на файловой системе
- быстрые фиксации работают в режиме `data=ordered`
- быстрая фиксация уменьшает задержку на выполнение `fsync` и `fdatasync` но только в случае если метаданные изменились
- запись в журнал быстрой фиксации выполняется процессом, вызвавшим `fsync`
- запись в обычный журнал выполняется потоком ядра
- быстрая фиксация работает не только с ext4, но и с другими журналируемыми файловыми системами

Быстрые фиксации изменений в журнале файловой системы ext4 (fast_commit)

- параметр commit (по умолчанию 5 секунд) задает частоту вызова sync на файловой системе
- быстрые фиксации работают в режиме `data=ordered`
- включение быстрых фиксаций:

```
root@tantor:~# dumpe2fs /dev/sda1 | grep Fast
dumpe2fs 1.47.0 (5-Feb-2023)
Fast commit length:       0
root@tantor:~# tune2fs -O fast_commit /dev/sda1
tune2fs 1.47.0 (5-Feb-2023)
root@tantor:~# dumpe2fs /dev/sda1 | grep Fast
dumpe2fs 1.47.0 (5-Feb-2023)
Fast commit length:       256
```

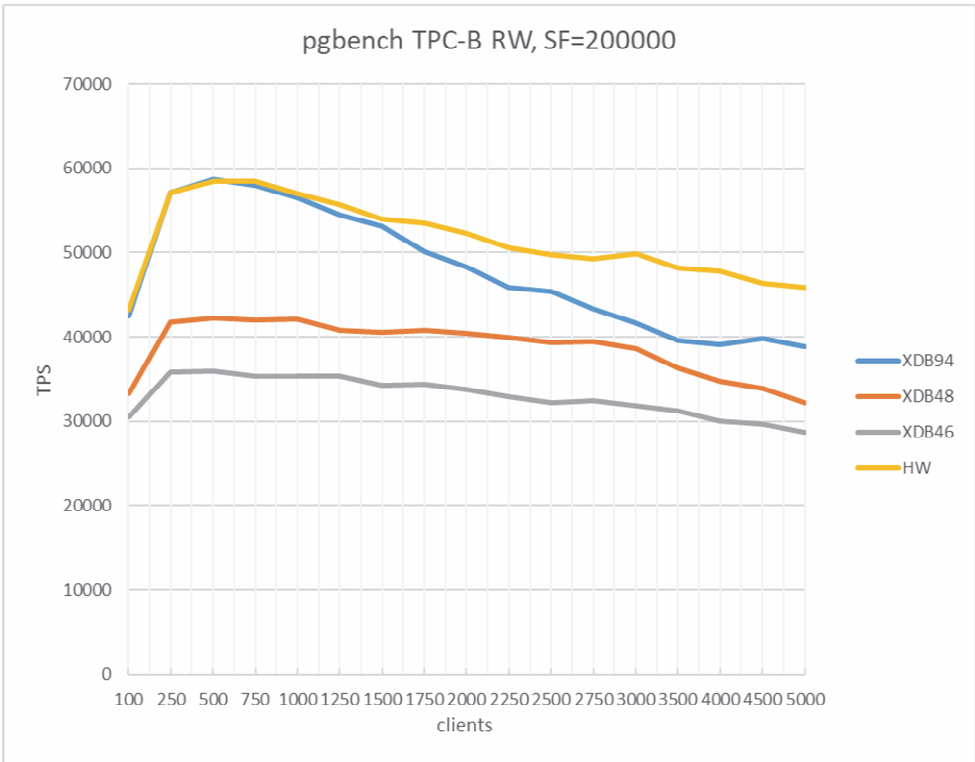
Утилита `pg_test_fsync`

- утилита командной строки, которая выполняет стандартные тесты для оценки производительности при выборе значения для параметра `wal_sync_method`

```
postgres@tantor:~$ 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                  3396.549 ops/sec     294 usecs/op
        fdatasync                      3459.610 ops/sec     289 usecs/op
        fsync                          3136.642 ops/sec     319 usecs/op
        fsync_writethrough                   n/a
        open_sync                      3275.082 ops/sec     305 usecs/op
Compare file sync methods using two 8kB writes:
```

Групповая фиксация транзакций

- транзакции могут фиксироваться группами
- обрабатывая COMMIT не обязательно что каждый серверный процесс выполнит запись в WAL-файл
- журнальная подсистема обычно не является узким местом при этом гарантирует отказоустойчивость
- алгоритм работы с WAL обеспечивает масштабируемость: при увеличении числа сессий до ~500 TPS растет



HW – физический сервер Tantor xData 96CPU, 1536GB RAM, 18TB SSD
shared_buffers=384GB

контейнер Tantor XDataBox
94CPU, 1120GB RAM, LVM 5TB,shared_buffers=384GB
48CPU, 560GB RAM, LVM 5TB, shared_buffers=192GB
46CPU, 560GB RAM, LVM 5TB, shared_buffers=192GB

Параметры `commit_delay` и `commit_siblings`

- параметр конфигурации устанавливает максимальную задержку в микросекундах, которую серверный процесс будет ждать после создания журнальной записи с `COMMIT` в буфере WAL перед вызовом команды записи в WAL-файл если имеется `commit_siblings` открытых транзакций и `fsync=on`
- по умолчанию значение ноль, задержки нет
- не ускоряет запись в WAL
- можно использовать для ограничения TPS чтобы не возникло узкое место в подсистеме отличной от WAL
- значение `commit_delay`, с которого можно начать - половина от задержки выдаваемой утилитой `pg_test_fsync` для используемого `wal_sync_method`
- изменение значений параметров `commit_delay` и `commit_siblings` не требует рестарта экземпляра

Команды шин ввода-вывода `discard/trim`

- важен для SSD
- `trim` - для SSD на шине SATA
- `discard` - для SSD на шине pci
- контроллер SSD не знает о том, что на нем располагаются файлы и есть файловые системы
- операционная система должна посылать команды очистки блоков (физических секторов) содержимое которых ей не нужно
- ресурс записи в режиме SLC ~100000 циклов
- ресурс записи не-SLC ~1000 циклов

Поддержка `discard/trim`

- если у контроллера не будет свободных блоков скорость записи и срок жизни SSD будут низкими
- контроллер SSD обычно очищает блоками размера `erase size`

```
root@tantor:~# tune2fs -o +discard /dev/sda1
root@tantor:~# tune2fs -l /dev/sda1 | grep discard
Default mount options:    user_xattr acl discard
root@tantor:~# fstrim -v /
fstrim: /: the discard operation is not supported
root@tantor:~# lsblk --discard
NAME    DISC-ALN DISC-GRAN DISC-MAX DISC-ZERO
sda            0        0B       0B         0
└─sda1         0        0B       0B         0
```

Рекомендации по использованию SSD

- не рекомендуется активно писать на SSD, если процент заполнения больше чем ~80% (зависит от алгоритма контроллера SSD)
- Для того чтобы часть SSD была всегда свободна можно создавать раздел меньшего размера, чем SSD
- статус службы, выполняющей discard(trim) по неиспользуемым блокам на файловых системах, использующие устройства с поддержкой discard(trim):

```
root@tantor:~# systemctl status fstrim.timer
  fstrim.timer - Discard unused blocks once a week
  Loaded: loaded (/lib/systemd/system/fstrim.timer; enabled; preset: enabled)
  Active: active (waiting) since ..; 7h ago
 Trigger: 1 day 6h left
Triggers: fstrim.service
   Docs: man:fstrim
systemd[1]: Started fstrim.timer - Discard unused blocks once a week.
```

Параметр `max_files_per_process`

- значение по умолчанию 1000
- по достижению значения серверный процесс будет часто закрывать-открывать файлы
- большое количество файлов может открываться если команды работают с большим количеством relations или таблицами состоящими из большого количества гигабайтных файлов

```
select setting, min_val,max_val,vartype, short_desc from pg_settings where
name = 'max_files_per_process';
 setting | min_val |  max_val   | vartype |          short_desc
---------+---------+------------+---------+------------------------------
 1000    | 64      | 2147483647 | integer | Sets the maximum number of
                     simultaneously open files for each server process.
```

Увеличение значения max_files_per_process

- при увеличении значения этого параметра нужно убедиться, что операционная система не ограничит число открытых файлов:

```
ERROR:  could not open relation 5/16550: Too many open files in system
```

- реальные ограничения запущенных процессов:

```
for PID in $(pgrep "postgres"); do cat /proc/$PID/limits | grep files; done | uniq
Max open files   1024     524288    files
```

- ограничения процессов запускаемых не через службы:

```
sudo -u postgres bash -c 'ulimit -n'
1024
```

- в файле `/etc/security/limits.conf` добавить строки:

```
postgres hard nofile infinity
postgres soft nofile infinity
```

- в файлах служб `/usr/lib/systemd/system/tantor*` добавить после `[Service]`

```
LimitNOFILE=infinity
LimitNOFILESoft=infinity
```

Временная файловая система (tmpfs)

- файловая система, использующая оперативную память для хранения файлов
- СУБД Tantor не использует `tmpfs` для хранения файлов
- не стоит использовать `tmpfs`

```
root@tantor:~# mount | grep tmp
udev on /dev type devtmpfs
tmpfs on /run type tmpfs
tmpfs on /dev/shm type tmpfs
tmpfs on /run/lock type tmpfs
ramfs on /run/credentials/systemd-tmpfiles-setup-dev.service type ramfs
ramfs on /run/credentials/systemd-tmpfiles-setup.service type ramfs
tmpfs on /run/user/102 type tmpfs
tmpfs on /run/user/1000 type tmpfs
```

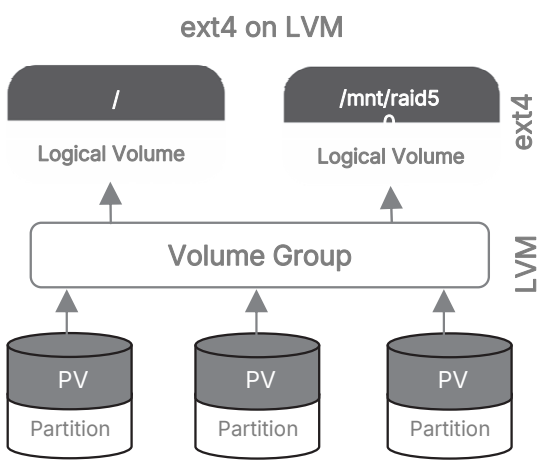
RAID

- массив "дисков" (устройств хранения)
- используется как том (единое блочное устройство)

LVM

- Logical Volume Manager (менеджер логических томов)
- объединяет физические диски в группы - Volume Groups (VG), на которых можно создавать логические разделы Logical Volumes (LV)
- можно изменять размеры LV и добавлять новые диски в VG без перемещения данных и изменения файловых систем
- LVM не увеличивает производительность, добавляет дополнительный уровень абстракции, что увеличивает вероятность ошибок при переконфигурировании
- используется при создании RAID 5+0:

```
mdadm --create /dev/md0 --level=5 --raid-devices=2 /dev/sd1 /dev/sd2
mdadm --create /dev/md1 --level=5 --raid-devices=2 /dev/sd3 /dev/sd4
pvcreate /dev/md0
pvcreate /dev/md1
vgcreate vg0 /dev/md0 /dev/md1
lvcreate -L 100%FREE -n lv0 vg0
mkfs.ext4 /dev/vg0/lv0
mkdir /mnt/raid50
mount /dev/vg0/lv0 /mnt/raid50
```



Практика

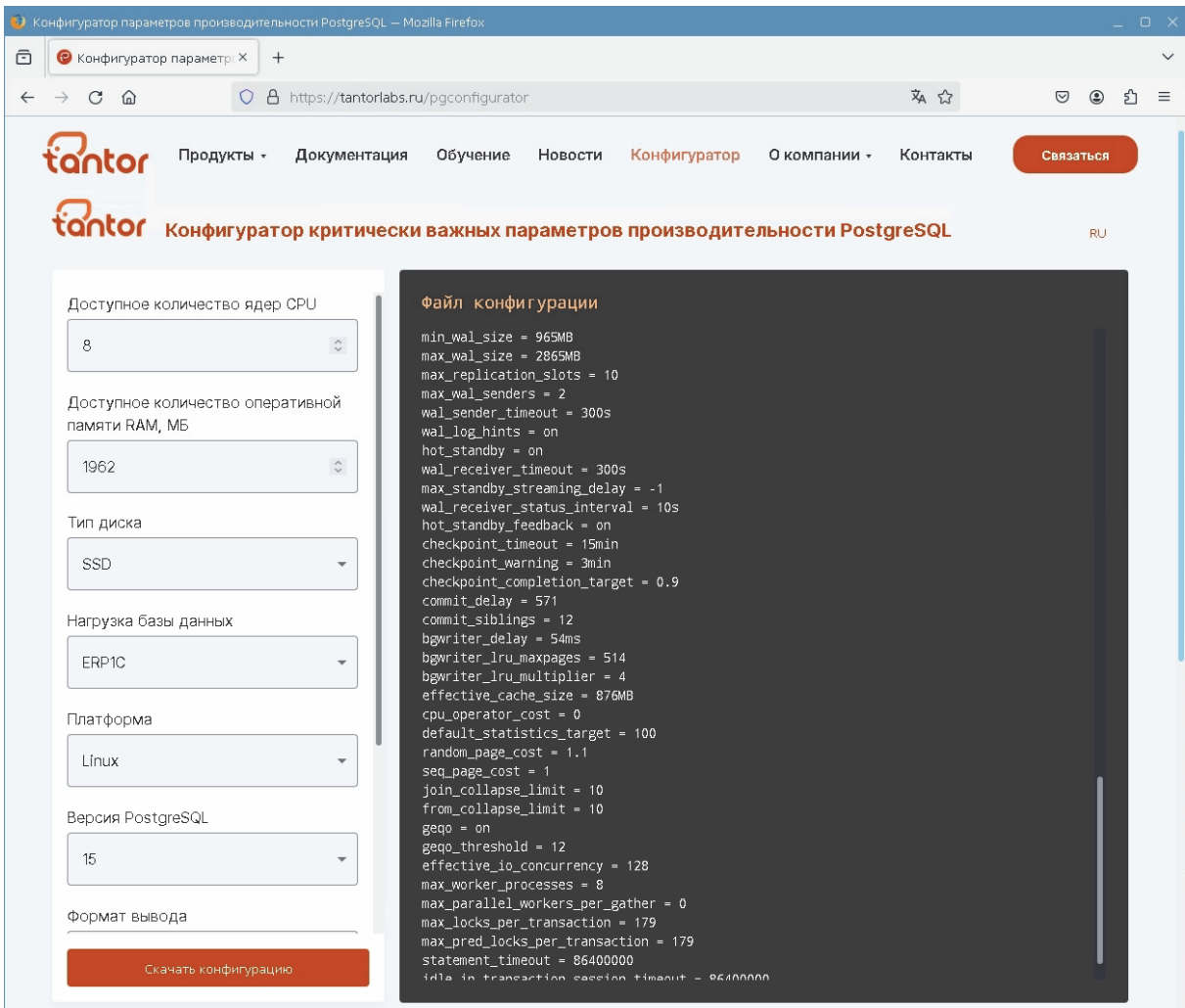
- Часть 1. Параметры дисковой подсистемы
- Часть 2. Установка пакетов в Astralinux
- Часть 3. Работа с SSD и тестирование производительности диска утилитой `fio`
- Часть 4. Тестирование журнала быстрой фиксации ext4
- Часть 5. Снятие ограничения на число открытых файлов
- Часть 6. Пример командных файлов для тестирования
- Часть 7. Пример создания программ для тестирования

5

Начальная настройка СУБД

Конфигураторы

- утилита pg_configurator
- веб-версия http://tantorlabs.ru/pgconfigurator
- вводятся характеристики хоста и планируемой нагрузки
- выдает параметры конфигурации



Параметры `shared_buffers`, `temp_buffers`, `effective_cache_size`

- `shared_buffers` по умолчанию 128MB. Можно установить в ~1/4 размера физической памяти. После изменения значения нужно перезапустить экземпляр
- `effective_cache_size` по умолчанию 4GB
  - дает планировщику оценку размера памяти, в которой могут поместиться блоки таблиц
  - влияет на расчет стоимости ввода-вывода, которая является часть общей стоимости плана выполнения
  - при небольшим значении параметра стоимость ввода вывода (cost) увеличивается для любых блоков - и таблиц и индексов и будут выбираться планы, в которых считывается меньше блоков
  - можно установить в 70-80% от размера физической памяти

Параметры work_mem, hash_mem_multiplier, maintenance_work_mem

- work_mem по умолчанию 4MB
  - Вместе с параметром hash_mem_multiplier влияет на память, выделяемую каждым серверным и параллельным процессом
  - Начальное значение устанавливается в зависимости от оценки количества сессий, в которых выполняются команды, обрабатывающие в памяти большие наборы данных, то есть зависит от типа нагрузки OLTP (однострочные вставки и выборки) и не-OLTP
- maintenance_work_mem значение по умолчанию 64MB
  - Задает объем памяти, выделяемый каждым процессом (серверным, параллельным), участвующем в выполнении команд VACUUM, ANALYZE, CREATE INDEX, ALTER TABLE ADD FOREIGN KEY
  - Начальное значение ~0.5% от размера физической памяти

Параметр `autovacuum_work_mem`

- по умолчанию -1 (используется значение `maintenance_work_mem`)
- выделяется сразу каждым рабочим процессом автовакуума
- процессы автовакуума не используют параллельные процессы
- под хранение tid (tuple identifier) используется не больше 1Гб, независимо от значения параметра `maintenance_work_mem`

Параметры `temp_file_limit` и `temp_tablespaces`

- `temp_file_limit` задаёт максимальный объём дискового пространства, который сможет использовать один процесс для временных файлов
- `temp_tablespaces` определяет названия табличных пространств в которых будут создаваться файлы временных объектов

Параметры `max_slot_wal_keep_size` и `transaction_timeout`

- `max_slot_wal_keep_size` по умолчанию -1 (без ограничений) максимальный размер журнальных файлов, который может оставаться в каталоге pg_wal после выполнения контрольной точки для слотов репликации
- `transaction_timeout` по умолчанию ноль (таймаут отключен). Позволяет отменить любую транзакцию или одиночную команду, длительность которой превышает указанный период времени, а не только простаивающую. Защищает от удержания горизонта базы данных и раздувания файлов (bloat)

Параметры `max_connections` и `client_connection_check_interval`

- `max_connections` устанавливает максимальное число одновременных подключений к экземпляру
  - по умолчанию 100
  - Изменение значения требует перезапуск экземпляра, поэтому стоит установить в первую очередь исходя из планируемого количества сессий.
  - Смежные параметры `reserved_connections` и `superuser_reserved_connections`
  - при увеличении значения сначала увеличивать на физических репликах, иначе они перестанут обслуживать запросы
- `client_connection_check_interval` по умолчанию ноль, проверка отключена. Позволяет прерывать длительные запросы, результат которых клиент не сможет получить.

Параметр `max_locks_per_transaction`

- число блокировок объектов и рекомендательных блокировок (advisory locks), которые могут использовать приложения, ограничено на экземпляре произведением `max_locks_per_transaction * (max_connections + max_prepared_transactions)`
- по умолчанию `max_prepared_transaction=0` и менять его не нужно если не используются распределенные транзакции
- формула устанавливает объем разделяемой памяти для хранения блокировок в памяти экземпляра
- утилита `pg_dump` с использованием параллельных процессов (параметр `--jobs`) устанавливает блокировки на все выгружаемые объекты
- для 300000 блокировок потребуется ~48Мб разделяемой памяти

Параметры `max_worker_processes` и `max_parallel_workers`

- `max_worker_processes` по умолчанию 8. Максимальное число фоновых (рабочих) процессов (background workers), которые могут могут быть запущены на экземпляре
  - устанавливается только на уровне кластера
- `max_parallel_workers` по умолчанию 8. Максимальное число параллельных процессов (parallel workers)
  - Фактически ограничено значением `max_worker_processes`, но может быть установлено в большее значение (для удобства)
  - Параллельные процессы используются для обслуживания команд SQL, выполняемых серверными процессами, а также для передачи изменений в логической репликации
  - можно установить на уровне базы, роли, роли в базе
- `max_parallel_maintenance_workers` по умолчанию 2. Максимальное число рабочих процессов, которые могут запускаться одной командой CREATE INDEX или VACUUM (без FULL)

Параметр `max_parallel_workers_per_gather`

- `max_parallel_workers_per_gather` (по умолчанию 2) задаёт максимальное число рабочих процессов, которые могут использоваться серверным процессом для обслуживания обычных (не maintenance) команд
- если число свободных процессов меньше расчетного, они и используются и команду обслуживает меньшее число процессов (степень параллелизма уменьшается).
- по умолчанию серверный процесс участвует в обработке данных (узлов плана ниже Gather) наравне с параллельными процессами, но обработает меньше строк, так как обслуживает последовательную часть плана и узлы Gather.
- участие серверного процесса в обработке параллельной части плана можно отключить `parallel_leader_participation=off`, но обычно не приводит к увеличению производительности

Параметры системы хранения

- `random_page_cost` значение по умолчанию 4 подходит только для для одиночного HDD. Для SSD значение должно быть близко к seq_page_cost, которое по умолчанию 1. Можно установить значение 1.1
- `effective_io_concurrency` значение по умолчанию 1 подходит только для для одиночного HDD. Для нескольких HDD (RAID) устанавливается в число устройств HDD по которым выполняется распределение записи (страйпинг). Для одиночного SSD на шине SATA подходят значения от 64. Для NVMe выше 128-512
- `maintenance_io_concurrency` по умолчанию 10. Используется вакуумом и анализом. Для SSD рекомендованные значения такие же как для `effective_io_concurrency`

Параметры контрольных точек

- параметр `full_page_writes=on` (по умолчанию)
- `checkpoint_completion_target=0.9` (по умолчанию)
- `checkpoint_timeout` рекомендуется установить в значение 20-30 минут
- `max_wal_size` размер журнальных файлов (pg_wal) по достижению которого вызывается контрольная точка раньше, чем установлено параметром `checkpoint_timeout`
  - допустимы при массовой загрузке или изменениям данных: когда блоки заполняются за секунды и позже не меняются
- статистика в столбцах `checkpoints_req` и `checkpoints_timed` представления `pg_stat_bgwriter`

Параметры процесса фоновой записи `bgwriter`

- `bgwriter_delay` по умолчанию 200ms, уменьшают до 20-40ms
- `bgwriter_lru_multiplier` по умолчанию 2. Обычно устанавливают значение больше 4-10.
- `bgwriter_lru_maxpages` по умолчанию 100 буферов, увеличивают до 400-500
- `bgwriter_flush_after` по умолчанию 512kB
- Значения устанавливают на основе данных из представления `pg_stat_bgwriter`:

```
select * from pg_stat_bgwriter \gx
-[ RECORD 1 ]---------+---------
checkpoints_timed     | 1461
checkpoints_req       | 90
checkpoint_write_time | 3805215
checkpoint_sync_time  | 116032
buffers_checkpoint    | 130655
buffers_clean         | 250785
maxwritten_clean      | 1421
buffers_backend       | 5243535
buffers_backend_fsync | 0
buffers_alloc         | 653441
```

Практика

- Часть 1. Блокировки объектов
- Часть 2. Наблюдение за памятью серверного процесса
- Часть 3. Временные таблицы и файлы
- Часть 4. Влияние параметров конфигурации на разделяемую память
- Часть 5. Параметр `max_connections` и производительность
- Часть 6. Размер кэша буферов и освобождение буферов

6

6-1

Структуры хранения

Таблицы

- объект в котором хранятся данные
- несколько видов: обычные таблицы (heap tables, строки хранятся неупорядоченно), нежурналируемые, временные, секционированные
- расширения могут создавать новые способы хранения данных и методы доступа к ним
- число и порядок следования столбцов задаются при создании таблицы
- после создания таблицы можно добавлять и удалять столбцы. При добавлении столбца он добавляется последним - после всех существующих столбцов
- можно поменять тип столбца

Служебные столбцы

- `xmin` - номер транзакции (xid), создавшей версию строки
- `xmax` - номер транзакции (xid), удаляющей или пытавшейся (транзакция не была зафиксирована по любой причине: вызван rollback, серверный процесс прерван) удалить строку или ноль
- `ctid` адрес физического расположения строки
- `tableoid` - oid таблицы, в которой физически содержится строка. Значения имеют смысл для секционированных и унаследованных таблиц
- `cmin` порядковый номер команды внутри транзакции начиная с нуля, создавшей версию строки
- `cmax` порядковый номер команды внутри транзакции начиная с нуля, удаляющей или пытавшейся удалить строку

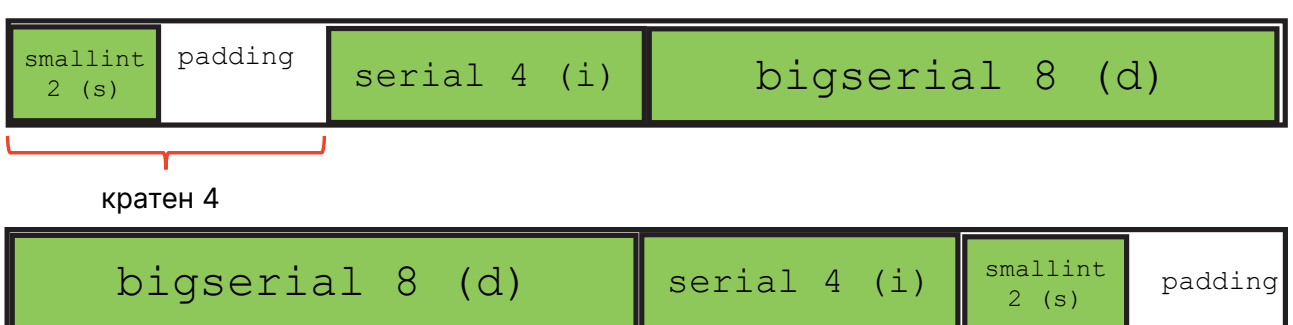
Расширение pageinspect

- стандартное расширение, не требует загрузки библиотеки
- контрольная сумма:
  - рассчитывается и сохраняется в момент записи в файл грязного блока
  - проверяется при считывании блока из файла в буфер кэша буферов
  - пока блок в буфере поле контрольной суммы в заголовке блока не меняется и не проверяется

```
create extension pageinspect;
drop table if exists t;
create table t(s text);
insert into t values ('a');
select * from page_header(get_raw_page('t', 0));
    lsn    |checksum|flags|lower|upper|special|pagesize|version|prune_xid
-----------+--------+-----+-----+-----+-------+--------+-------+---------
6/B6EC12C0|       0 |   0 |  28 | 8144|  8176 |   8192 |     5 |       0
vacuum full t;
select * from page_header(get_raw_page('t', 0));
    lsn    |checksum|flags|lower|upper|special|pagesize|version|prune_xid
-----------+--------+-----+-----+-----+-------+--------+-------+---------
6/B6ED5458| -11261 |   0 |  28 | 8144|  8176 |   8192 |     5 |       0
```

Padding и aligning

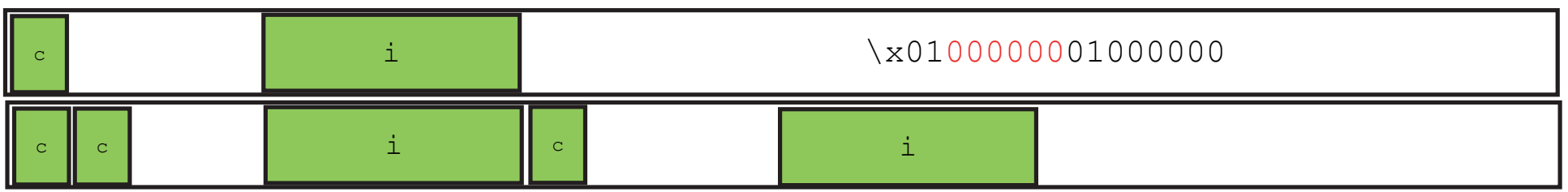
- выравнивание (align) - длинна поля кратна числу байт
- указано в столбцах pg_type.typalign и pg_attribute.attalign
- возможны следующие значения выравнивания:
  - s (short) 2 байта
  - i (int) 4 байта
  - d (double) 8 байт
  - c (char) без выравнивания (побайтово)
- padding - добавление неиспользуемого места, чтобы выполнить выравнивание



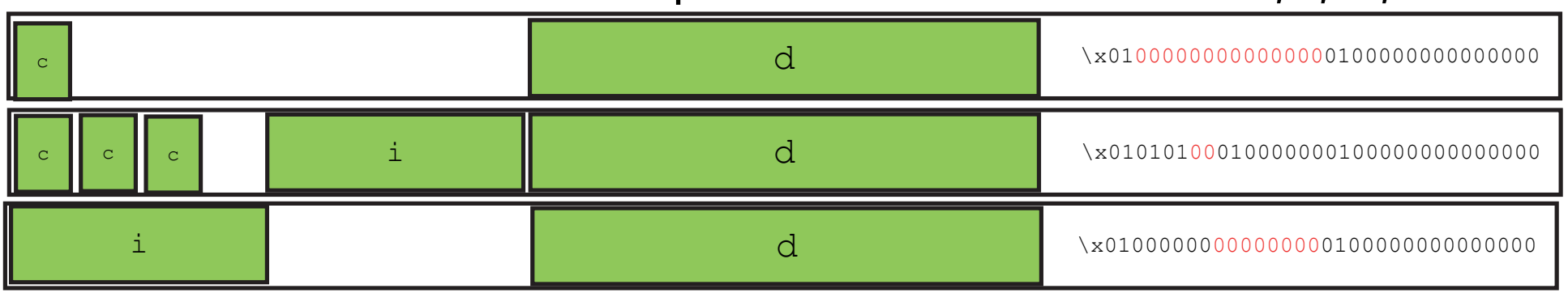
Aligning (выравнивание)

- "c" и типы переменной длинны до 127 байт не выравниваются:

- "i" - начало поля может располагаться только в 1,5,9,13... байте:

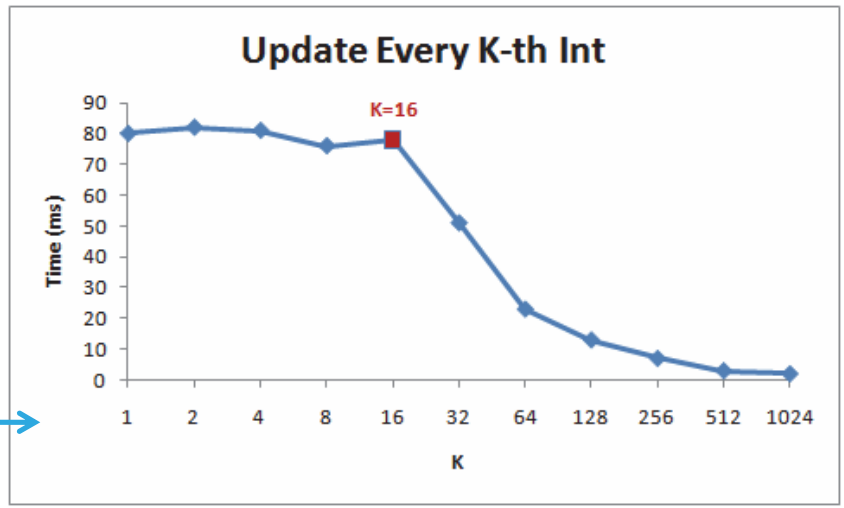


- "d" - начало поля может располагаться только в 1,9,17,25... байте:

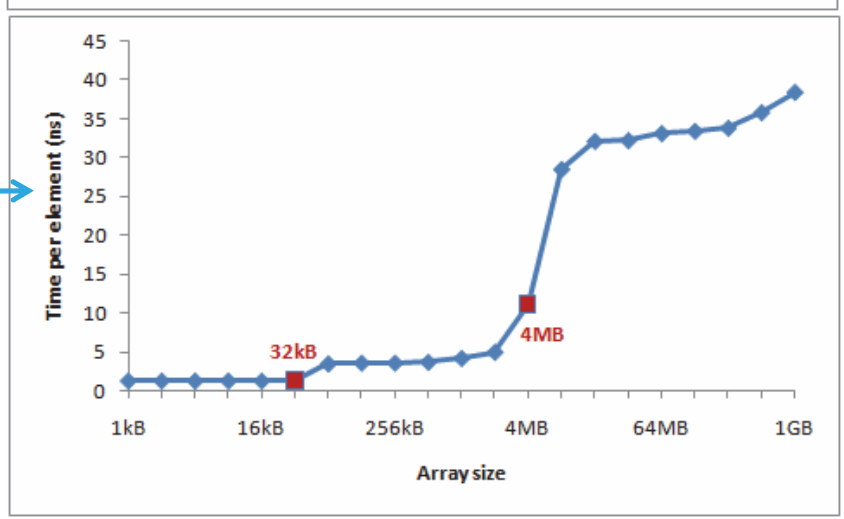


cache line

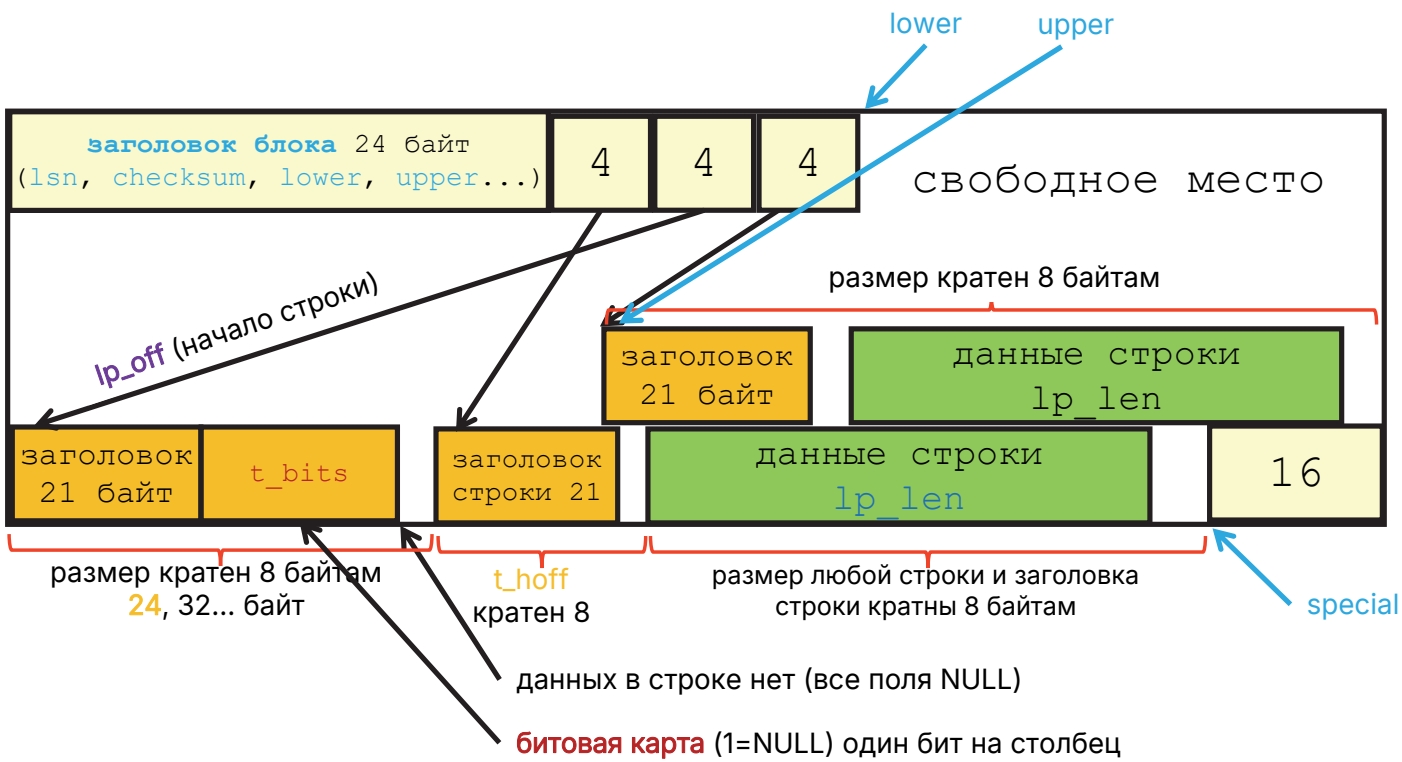
- выравнивание существенно ускоряет обработку данных, поэтому используется по возможности везде где это не приводит к сильному увеличению места хранения
- до 16*8=64 байт (K=16) время на обращение к памяти одинаково



- время на обработку данных, если они помещаются в кэше



Структура блока данных




```
select * from page_header
(get_raw_page('t','main',0));
-[ RECORD 1 ]---------
lsn        | 0/110DCF10
checksum   | 0
flags      | 0
lower      | 928
upper      | 944
special    | 8176
pagesize   | 8192
version    | 5
prune_xid | 0
```

```
select * from heap_page_items(get_raw_page('t','main',0)) limit 1;
lp|lp_off|lp_flags|lp_len|t_xmin|t_xmax|t_field3|t_ctid|t_infomask2|t_infomask|t_hoff| t_bits |t_oid
--+------+--------+------+------+------+--------+------+-----------+----------+------+--------+-----
18|   776|       1|    38| 43339|    0 |    369 |(0,18)|         6 |     2307 |   24 |01111100|
```

Число строк в блоке

- не больше 291 для строк длинной ноль (все поля пустые)
- максимальное число непустых строк, которые смогут поместиться в блок: **226, 185, 156 (157), 135 (136), 119(120), 107, 97, 88, 81, 75, 70, 65,** ... так как длинна строки кратна 8 байтам
- на каждую строку к заголовку блока добавляется 4 байта
- в начале блока 24 байта заняты служебными данными
- в Tantor SE, SE1C в конце блока таблиц используется 16 байт под служебные данные
- если столбцов в таблице не больше 8, то заголовок строки 24 байта
- заголовок строки кратен 8 байт может иметь размер 24, 32,... байт
- в заголовке строки выделяется место под битовую карту пустых значений в полях строки, если столбцы могут содержать NULL
- пустые поля не занимают места в области данных
- NULL не занимают места в области данных, использование NULL эффективно с точки зрения компактности хранения
- FILLFACTOR (процент заполнения) для таблиц по умолчанию 100%

Порядок следования столбцов в таблице

- первыми столбцами стоит делать d (выравниваются по 8 байт), затем i, затем s (smallint), затем c (boolean, char, uuid)
- типы переменной длинны (text, bytea, numeric и много других) имеют выравнивание i или d. Однако, **поля таких типов будут выравниваться только, если в них хранится больше 126 байт**
- у типов переменной длинны в столбце typlen таблицы pg_type отрицательное число: -1 обозначает тип varlena
- столбцы с большим количеством значений NULL стоит помещать после полей фиксированной ширины и до varlena (если в них будет храниться больше 126 байт)

Порядок следования столбцов и производительность

- столбцы, входящие в PRIMARY KEY должны идти первыми как не имеющие пустых значений и наиболее часто используемые
- дальше поля с типами данных фиксированной ширины, в которых нет пустых значений
- дальше столбцы с типами данных фиксированной ширины, в которых редко встречаются пустые значения
- столбцы, реже встречающиеся в запросах должны идти после столбцов, которые встречаются чаще
- столбцы с полями большего размера лучше делать последними
- учитывать алгоритм выноса полей в TOAST
- **если в таблице не больше 8 столбцов, то заголовок строки 24 байта**
- Если столбцов больше 8, то NULL в любом из полей строки увеличивает заголовок строки на 8 байт, но с другой стороны область данных уменьшается на размер поля

Практика

- Часть 1. Карта свободного пространства
- Часть 2. Изменение порядка следования столбцов
- Часть 3. Содержимое блоков таблицы
- Часть 4. Выравнивание полей в строках таблиц
- Часть 5. Выравнивание строк в блоках таблиц
- Часть 6. Хранение пустых (NULL) значений в строках таблиц
- Часть 7. Число строк в блоке таблицы

7

Индексы

Методы доступа к строкам

```
\dA
List of access methods
   Name    | Type
-----------+-------
 bloom     | Index
 brin      | Index
 btree     | Index
 columnar  | Table
 gin       | Index
 gist      | Index
 hash      | Index
 heap      | Table
 spgist    | Index
(9 rows)
```

- два типа "методов" (способов) доступа к строкам таблиц: табличный и индексный
- Методы доступа можно добавлять расширениями:
  - `create extension pg_columnar;`
  - `create extension bloom;`
- Табличные методы доступа определяют способ хранения данных в таблицах и обычно считывают все строки
- Индексные методы обычно считывают часть блоков таблицы
- Для индексных методов (способов) нужно создать вспомогательный объект, называемый индексом
- Индексы создаются на один или несколько столбцов таблицы:

```
create table t (id int8, d date, s text);
create index t_idx on  t using btree (id int8_ops, d date_ops);
create index t_idx1 on t using btree (s text_ops);
create index if not exists t_idx2 on t (id, d);
```

Класс операторов для индекса

- при создании индекса можно указать класс операторов отдельно для каждого столбца индекса

```
create table t (id int8, s text);
create index t_idx on t using btree (id int8_ops, s text_pattern_ops);
```

- если не указать класс операторов, то используется класс по умолчанию для типа столбца:

```
\dAc+ btree integer
                                  List of operator classes
  AM   | Input type | Storage type | Operator class | Default? | Operator family |  Owner
-------+------------+--------------+----------------+----------+-----------------+----------
 btree | integer    |              | int4_ops       | yes      | integer_ops     | postgres
```

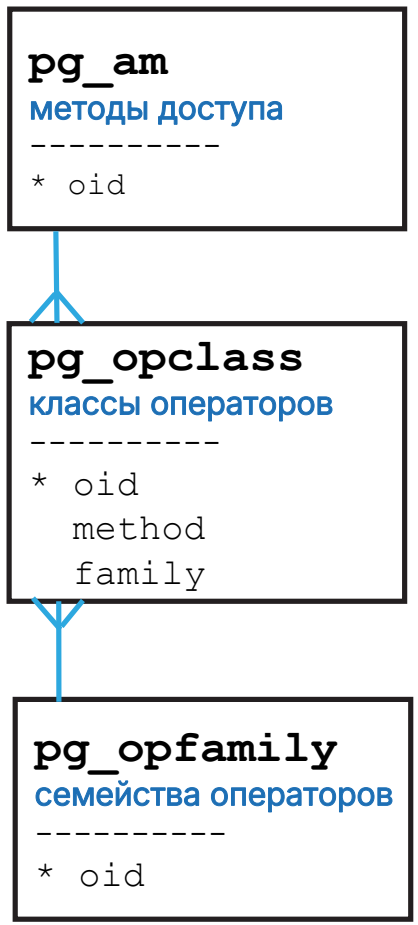
- список функций, используемых семействами операторов:

```
\dAp+ btree integer_ops
List of support functions of operator families
  AM   | Operator family | Registered left type | Registered right type | Number |  Function
-------+-----------------+----------------------+-----------------------+--------+----------------
btree  | integer_ops     | bigint               | bigint                |      1 | btint8cmp(bigint,bi..
btree  | integer_ops     | bigint               | bigint                |      2 | btint8sortsupport(in..
```

Семейства и классы операторов

- класс операторов связывает операторы, которые будут играть роли (стратегии) в методах которые использует логика индекса для упорядочивания (сравнения, сортировки, измерения расстояний, ассоциаций) данных. В классе операторов указаны названия "опорных" (supporting) функций, которые будут задействовать метод индекса при поиске или сортировке данных
- список операторов:

```
\dAo btree
  AM   | Operator family |     Operator      | Strategy | Purpose
-------+-----------------+-------------------+----------+---------
 btree | integer_ops     | <(bigint,bigint)  |        1 | search
 btree | integer_ops     | <=(bigint,bigint) |        2 | search
 btree | integer_ops     | =(bigint,bigint)  |        3 | search
 btree | integer_ops     | >=(bigint,bigint) |        4 | search
 btree | integer_ops     | >(bigint,bigint)  |        5 | search
...
```



Поддерживающие функции для индекса

- Для индексации методом btree:
  - достаточно чтобы в классе операторов имелась функция (BTORDER_PROC) сравнения (compare) двух значений
  - для эффективной сортировки (ORDER BY) желательно наличие второй функции быстрой сортировки значений (BTSORTSUPPORT_PROC)
  - для возможности использования планировщиком индекса в выражениях "RANGE" оконных функций нужна третья функция (BTINRANGE_PROC)
  - для поддержки дедупликации нужна четвертая функция (BTEQUALIMAGE_PROC).
- Список функций:

```
\dAp+ btree integer_ops
List of support functions of operator families
  AM   | Operator family | Registered left type | Registered right type | Number |  Function
-------+-----------------+----------------------+-----------------------+--------+-----------------
btree  | integer_ops     | bigint               | bigint                |      1 | btint8cmp(bigint,bi..
btree  | integer_ops     | bigint               | bigint                |      2 | btint8sortsupport(in..
btree  | integer_ops     | bigint               | bigint                |      3 | in_range(bigint,..
btree  | integer_ops     | bigint               | bigint                |      4 | btequalimage(oid)
```

Индексы для ограничений целостности

- для ограничений целостности PRIMARY KEY и UNIQUE необходим **уникальный** индекс **типа btree** со столбцами, входящими в ограничение целостности.
- остальные индексы могут использоваться для ускорения запросов ("аналитические индексы"), полнотекстового поиска
- индексы ускоряют поиск строк и замедляют добавление, изменение, удаление строк
- индексы используют место на диске, размер сопоставим с размером таблицы
- пример замены индекса другим индексом:

```
create table t3 (n int4 primary key, m int4);
Indexes:
    "t3_pkey" PRIMARY KEY, btree (n)
create unique index concurrently t3_pkey1 on t3 (m,n);
ALTER TABLE t3 DROP CONSTRAINT t3_pkey, ADD CONSTRAINT t3_pkey PRIMARY KEY USING INDEX t3_pkey1;
NOTICE:  ALTER TABLE / ADD CONSTRAINT USING INDEX will rename index "t3_pkey1" to "t3_pkey"
Indexes:
    "t3_pkey" PRIMARY KEY, btree (m, n)
```

Индекс типа btree

- Для числового столбца (int2, int4, int8) во внутреннем (промежуточном) блоке при 70% заполнения помещается 286 ссылок, при 100% поместится 407 ссылок
- Если в индексе указать больше одного столбца:
  - порядок индексируемых столбцов важен
  - можно представить, что индексируется результат конкатенации значений столбцов
  - первый столбце называется ведущим
  - индекс называется составным
- Индексная запись должна поместиться в 1/3 блока (2704 байта):

```
create table t (id int8, s text storage plain);
create index t_idx on t (s text_ops) include (id);
insert into t values (1, repeat('a',2700));
ERROR:  index row size 2720 exceeds btree version 4 maximum 2704 for index "t_idx"
DETAIL:  Index row references tuple (0,2) in relation "t".
HINT:  Values larger than 1/3 of a buffer page cannot be indexed.
Consider a function index of an MD5 hash of the value, or use full text indexing.
```

Функции расширения pageinspect для btree

- `bt_metap(relname)` выдаёт информацию из блока метаданных индекса. Это всегда первый блок первого файла индекса (блок номер ноль)
- `bt_page_stats(..)` и `bt_multi_page_stats(..)` номера соседних блоков слева (btpo_prev) и справа (btpo_next) на том же уровне
- `bt_page_items(..)` содержимое блока индекса

```
CREATE TABLE t(s text storage plain) with (autovacuum_enabled=off, fillfactor=40);
create index t_idx on t (s);
INSERT INTO t VALUES (repeat('a',2500));INSERT INTO t VALUES (repeat('b',2500));
INSERT INTO t VALUES (repeat('c',2500));INSERT INTO t VALUES (repeat('d',2500));
select * from bt_multi_page_stats('t_idx',1,-1);
blkno|type|live_items|dead_items|avg_item_size|pagesize|freesize|btpo_prev|btpo_next|btpo_level|btpo_flags
-----+----+----------+----------+-------------+--------+--------+---------+---------+----------+----------
   1 | l  |        2 |        0 |        2512 |   8192 |   3116 |       0 |       2 |        0 |        1
   2 | l  |        3 |        0 |        2512 |   8192 |    600 |       1 |       0 |        0 |        1
   3 | r  |        2 |        0 |        1260 |   8192 |   5620 |       0 |       0 |        1 |        2
select itemoffset, ctid, itemlen, nulls, vars, dead, htid, tids, data from bt_page_items('t_idx',1);
 itemoffset | ctid  | itemlen | nulls | vars | dead | htid  | tids |          substring
------------+-------+---------+-------+------+------+-------+------+-----------------------
          1 | (1,1) |    2512 | f     | t    |      |       |      | 20 27 00 00 62 62 62 62
          2 | (0,1) |    2512 | f     | t    | f    | (0,1) |      | 20 27 00 00 61 61 61 61
```

Индексы с дедупликацией в листовых блоках

- если индекс не уникальный, то при большом количестве дубликатов они хранятся компактно за счет дедупликации в листовых блоках
- не все типы данных поддерживают дедупликацию
  - не поддерживают: `numeric, jsonb, float4, float8`
  - не поддерживают: массивы, составные, диапазонные типы
  - индексы INCLUDE не поддерживают дедупликацию
- значения проиндексированных столбцов хранятся в записи индекса, а ссылки на строки таблицы хранится в d в столбце tid (tuple id, идентификатор строк таблицы) в виде отсортированного массива ctid
- пример двух записей с дедупликацией и одной без дедупликации:

```
select itemoffset, ctid, itemlen, nulls, vars, dead, htid, data, tids[0:3] from bt_page_items('td_idx',1);
 itemoffset|  ctid   |itemlen|nulls|vars|dead|htid   |          data           |        tids
-----------+---------+-------+-----+----+----+-------+-------------------------+-----------------------
         1 |(16,8414)|  1352 | f   | f  | f  |(0,1)  |01 00 00 00 00 00 00 00| {"(0,1)","(0,2)","(0,3)"
         2 |(16,8377)|  1128 | f   | f  | f  |(10,13)|01 00 00 00 00 00 00 00| {"(10,13)","(10,14)","(1
         3 |(19,9)   |    16 | f   | f  | f  |(19,9) |01 00 00 00 00 00 00 00|
```

Проверка поддерживается ли дедупликация

- дедупликация не поддерживается с типами данных: `numeric, jsonb, float4, float8`, массивами, составными, диапазонными типами
- индексы со столбцами INCLUDE не поддерживают дедупликацию
- примеры запроса для проверки поддерживается ли дедупликация:

```
create table td(n timestamp, n1 date, n2 integer, n3 char, n4 text, n5 varchar);
create index td_idx on td (n,n1,n2,n3,n4,n5);
select allequalimage from bt_metap('td_idx');
 allequalimage
---------------
 t
create index td1_idx on td (n) include (n1);
select allequalimage from bt_metap('td1_idx');
 allequalimage
---------------
 f
create table td(id int8[]);
create index td_idx on td (id);
select allequalimage from bt_metap('td_idx');
 allequalimage
---------------
 f
```

Параметры создания индекса и их влияние на производительность

- по умолчанию индекс строиться в возрастающем порядке
- при создании индекса можно указать обратный порядок: DESC
- правые блоки оптимизированы для вставок
- по умолчанию пустые значения сохраняются в правых блоках
- вставки строк с NULL замедлятся при использовании NULLS FIRST
- вакуум читает все блоки индекса в физическом порядке от первого до последнего блока
- включение (include) в индекс значений столбцов не включая их в ключевые значения увеличивает размер индекса

```
create unique index concurrently if not exists t1_idx1 ON t1 using btree
(c2 desc nulls first, upper(c1) ) include (c3,c4) with (fillfactor=100,
deduplicate_items=off) WHERE c2>0;
```

Частичные (partial) индексы

- создаются по части строк таблицы
- предикат WHERE указывается при создании индекса и определяет индексируемые строки
- полезны тем, что позволяют избежать индексирования наиболее часто встречающихся значений
- частичный индекс может быть уникальным
- размер частичного индекса обычно меньше
- пример создания частичного индекса:

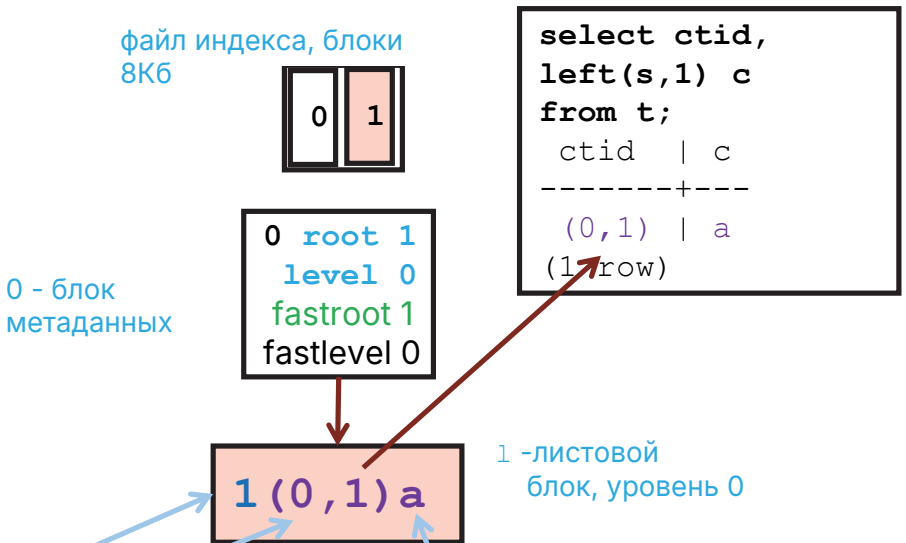
```
create unique index t1_idx1 ON t1 (c2 desc nulls first, upper(c1))
include (c3,c4) WHERE c2>0;
```

Эволюция индексов: создание, удаление, перестройка

- команды `create/drop/reindex index имя_индекса` устанавливают блокировку SHARE, не совместимую с внесением изменений в строки таблицы
- эти команды могут выполняться одновременно, они совместимы сами с собой, при этом с concurrently не совместимы
- автовакуум не совместим ни с `concurrently`, ни без
- для временных индексов на временные таблицы не надо использовать concurrently, так как блокировок на временные объекты нет
- `create/reindex concurrently` сканирует таблицу два раза, без `concurrently` один раз
- `concurrently` позволяет выполняться командам `SELECT, WITH, INSERT, UPDATE, DELETE, MERGE` и позволяет использовать быстрый путь (fastpath) блокирования объектов (таблиц, индексов, секций)

Структура индекса типа btree

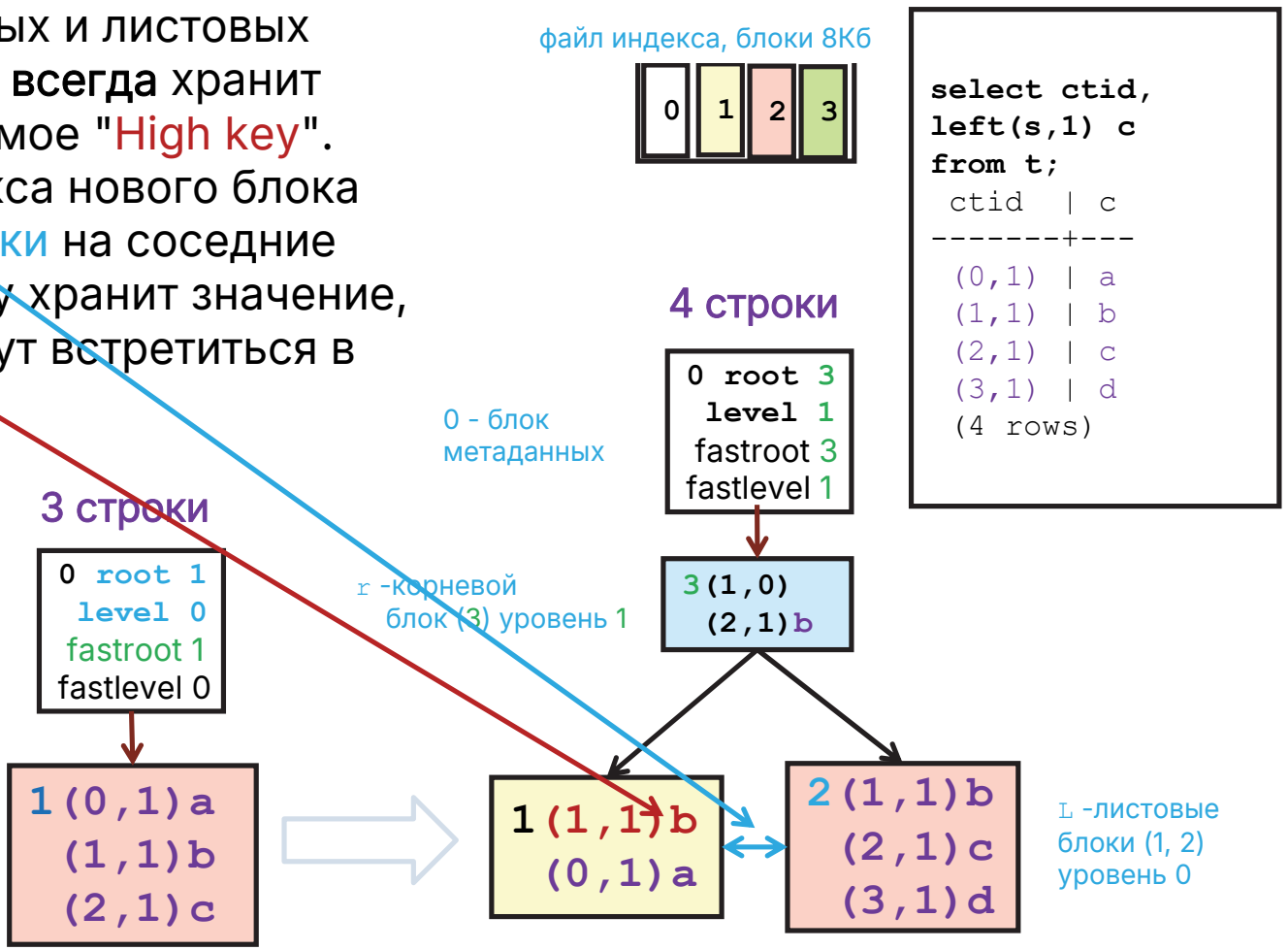
- типы блоков: метаданных, корневой, внутренние, листовые
- путь от корня до листовых блоков одинаковой глубины (балансирован)
- в листовых блоках ссылки на строки таблицы (блок, строка в блоке)
- в не-листовых блоках ссылки на блоки индекса (блок, строка в блоке)



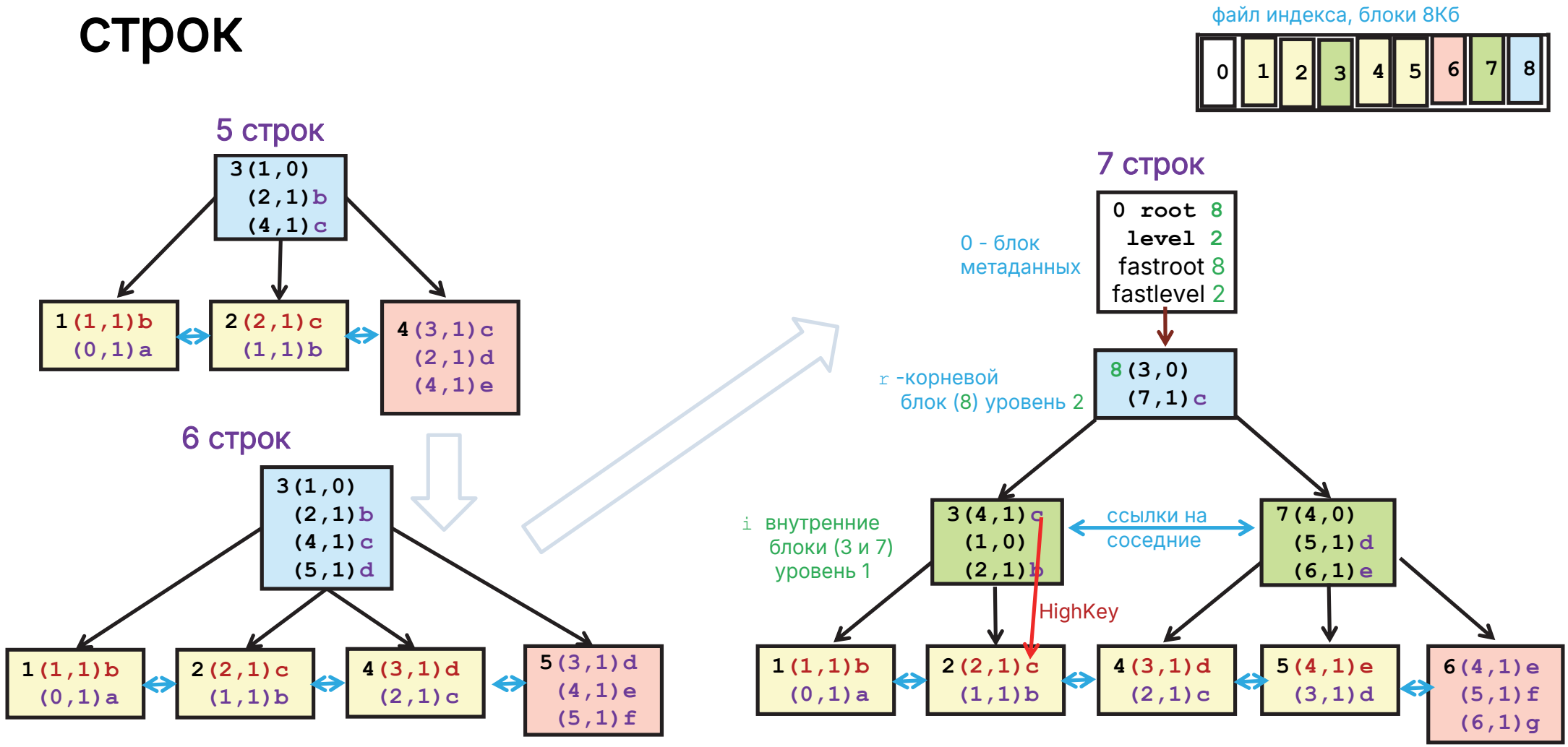

```
create table t(s text storage plain) with (fillfactor=10);
create index t_idx on t (s) with (fillfactor=10);
insert into t values (repeat('a',2500));
select itemoffset, ctid, itemlen, nulls, vars, dead, htid, tids, left(data,24) data,
chr(nullif(('0x0'||substring(data from 13 for 2))::integer,0)) c from bt_page_items('t_idx',1);
 itemoffset| ctid  | itemlen | nulls | vars | dead | htid  | tids |            data          | c
 ----------+-------+---------+-------+------+------+-------+------+--------------------------+--
         1 | (0,1) |    2512 | f     | t    | f    | (0,1) |      | 20 27 00 00 61 61 61 61 | a
```

High Key в структуре индекса

- первая строка в промежуточных и листовых блоках кроме самых "правых" **всегда** хранит служебное значение, называемое "High key".
- при вставке в структуру индекса нового блока обновляются High keys и ссылки на соседние блоки того же уровня. High key хранит значение, большее, чем те, которые могут встретиться в этом блоке.

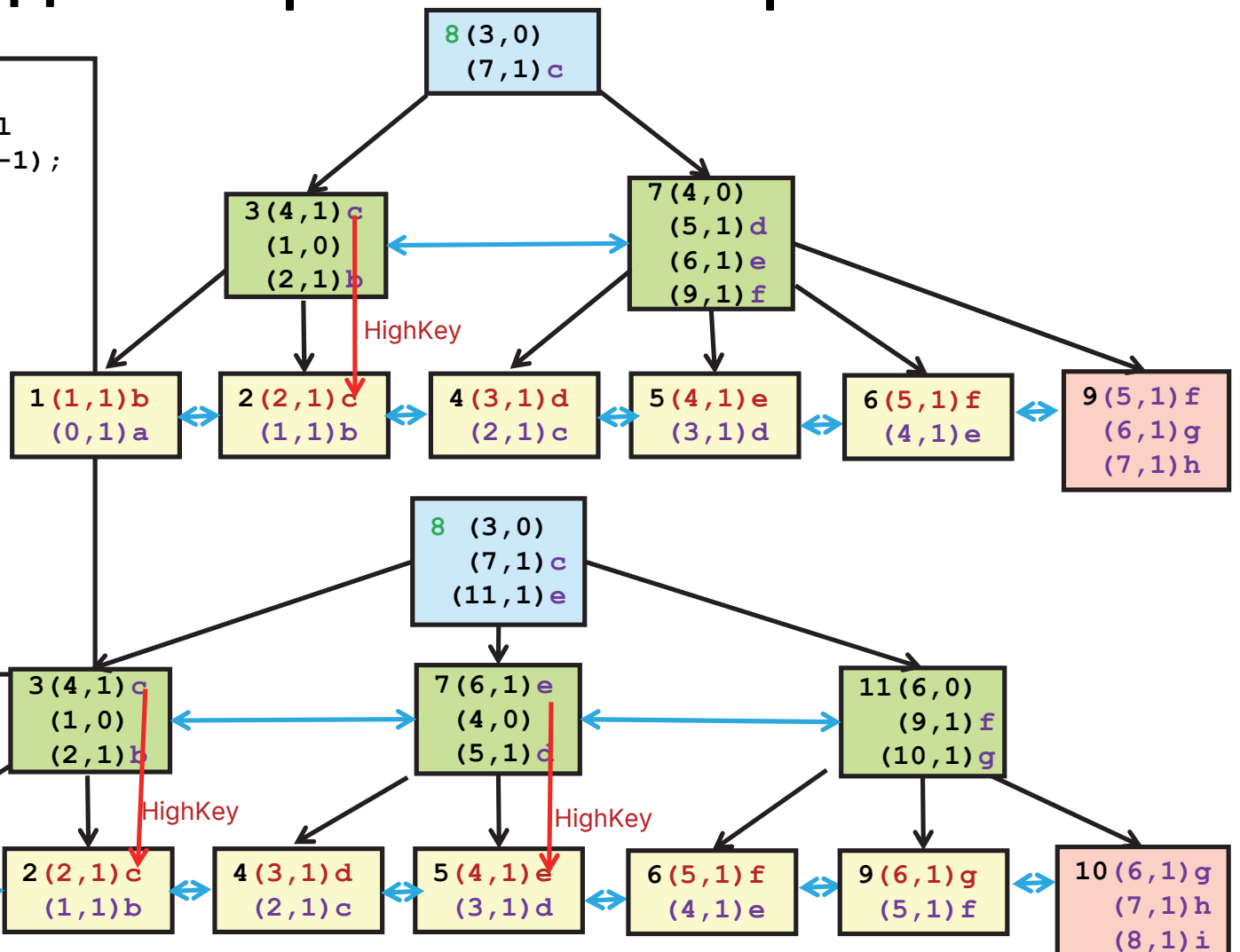


Изменение структуры индекса при добавлении строк



Пример роста индекса при вставке строк

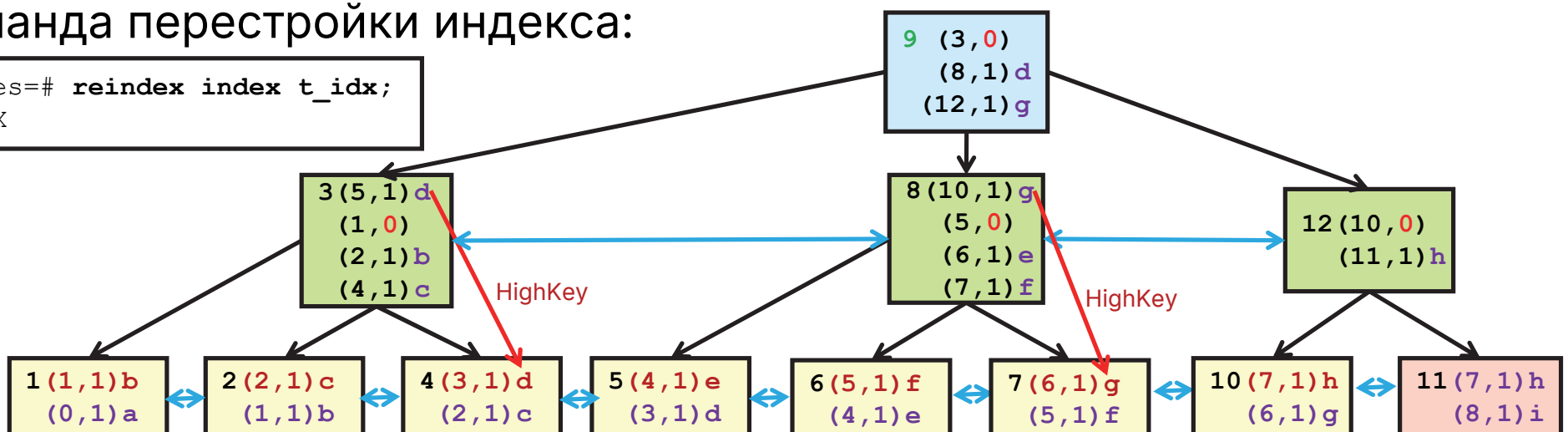
```
 select blkno blk, type, live_items live,
btpo_prev prev, btpo_next next, btpo_level
level from bt_multi_page_stats('t_idx',1,-1);
 blk | type | live | prev | next | level
-----+------+------+------+------+------
   1 | l    |    2 |    0 |    2 |     0
   2 | l    |    2 |    1 |    4 |     0
   3 | i    |    3 |    0 |    7 |     1
   4 | l    |    2 |    2 |    5 |     0
   5 | l    |    2 |    4 |    6 |     0
   6 | l    |    2 |    5 |    9 |     0
   7 | i    |    3 |    3 |   11 |     1
   8 | r    |    3 |    0 |    0 |     2
   9 | l    |    2 |    6 |   10 |     0
  10 | l    |    3 |    9 |    0 |     0
  11 | i    |    3 |    7 |    0 |     1
(11 rows)
```



Структура индекса после его перестроения

- HighKey вышестоящего блока должно присутствовать в самом правом нижестоящем блоке, на который есть ссылка в вышестоящем блоке. Если значение отличается, то процесс должен прочесть блоки нижнего уровня правее самого правого, так как скорее всего он уже не является правым
- в самых правых блоках на своём уровне HighKey не хранятся, так как правее их блоков нет и проверка не нужна
- команда перестройки индекса:

```
postgres=# reindex index t_idx;
REINDEX
```

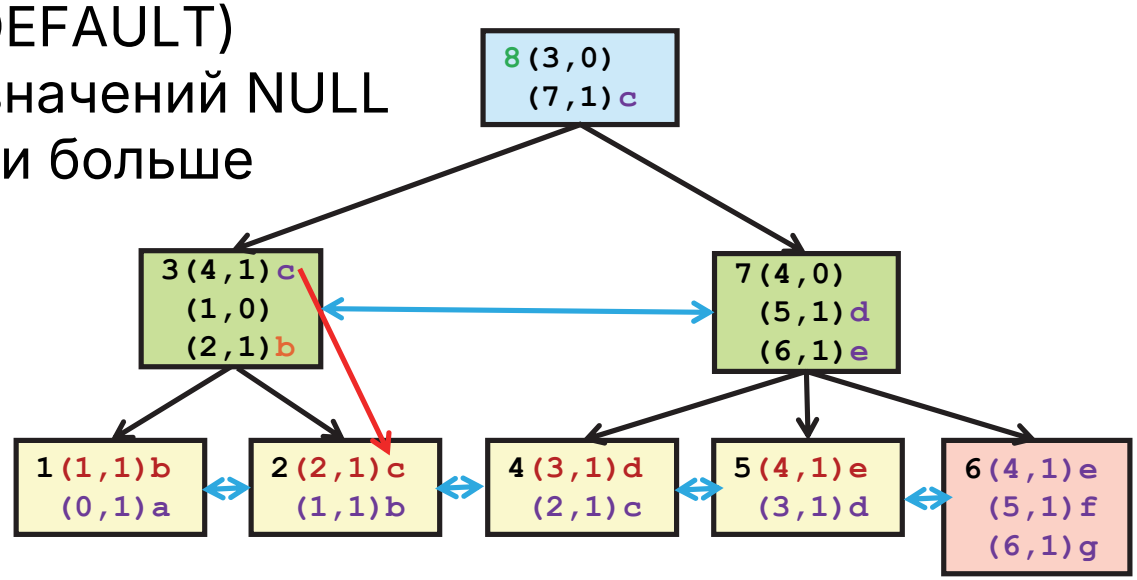


FILLFACTOR в индексах типа btree

- для индексов на столбец, заполняемый возрастающей последовательностью стоит устанавливать fillfactor=100
- в случае установки `fillfactor=100` левый блок при делении самого правого листового блока будет максимально заполнен, данные будут храниться более компактно, деление правого блока будет происходить реже
- в btree индексах fillfactor установлен в значения:
  - 90% для листовых блоков
  - 70% для нелистовых блоков и это не меняется
  - 96% при разделении любого листового блока, который полностью заполнен дубликатами (одним и тем же значением)
- для листовых блоков fillfactor применяется:
  - во время построения индекса
  - при разделении крайней правой страницы как листового, так и промежуточных уровней

Быстрый путь (fastpath) вставки в индексы

- процесс, который выполнил вставку в правый листовой блок запоминает ссылку на него и при последующей вставке, если новое значение больше предыдущего (или пусто) и не проходит путь от корня до листового блока
- ускоряет вставки:
- в целочисленные столбцы, заполняемые последовательностью
- датавременные (заполняемые DEFAULT)
- при вставке преимущественно значений NULL
- используется при числе level=2 и больше



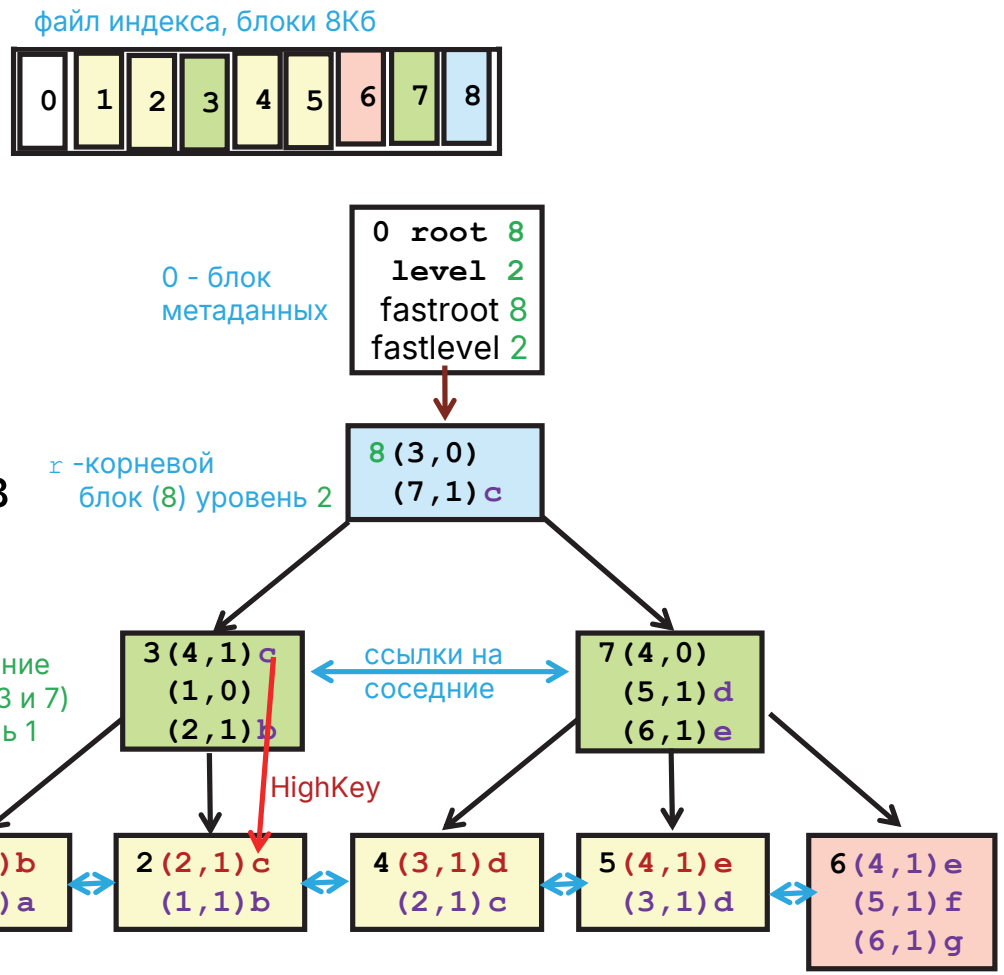
Внутристраничная очистка в индексах

- выполняется при индексном сканировании
- если строка в таблице удалена, индексная запись на строку или цепочку версий помечается флагом LP_DEAD
- пометка может ставиться командой SELECT при индексном сканировании
- журнальная запись не создается, но блок грязнится
- помеченная индексная запись будет очищена при выполнении команд, которые меняют данные в таблице

```
create table t (id int primary key, c text) with (autovacuum_enabled = off);
insert into t SELECT i, 'simple delete ' || i from generate_series(1, 10000) as i;
delete from t where id between 100 and 9000;
analyze t;
explain (analyze, settings, buffers, costs off) select * from t where id between 1 and 9000;
select itemoffset, ctid, itemlen, nulls, vars, dead, htid, tids, left(data,8) from
bt_page_items('t_pkey',20) where dead=true limit 2;
 itemoffset |  ctid   | itemlen | nulls | vars | dead |  htid   | tids | substring
------------+---------+---------+-------+------+------+---------+------+-----------
          2 | (42,37) |      16 | f     | f    | t    | (42,37) |      | bd 19 00
          3 | (42,38) |      16 | f     | f    | t    | (42,38) |      | be 19 00
```

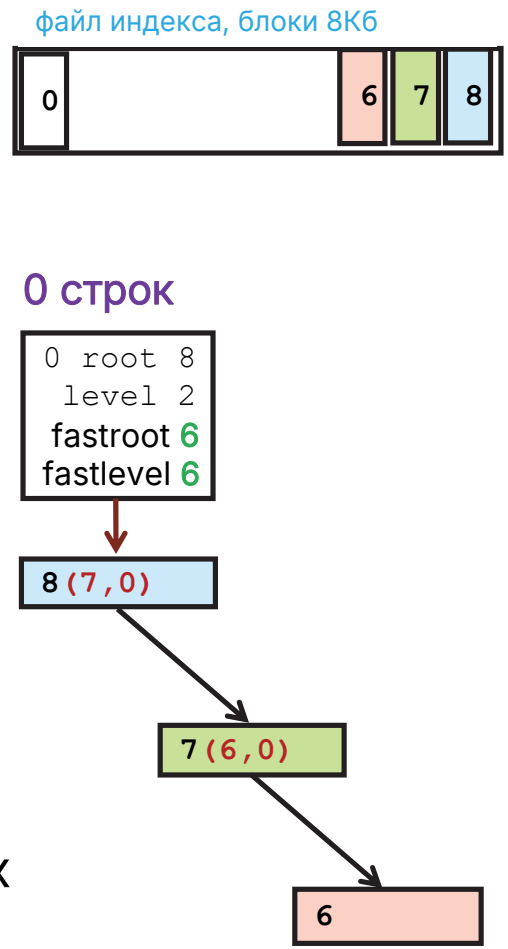
Влияние удаления строк на индексы

- при удалении строки в таблице индексная запись не удаляется, так как строка в табличном блоке остаётся до вакуумирования
- удалять из индексов записи может вакуум и внутристраничная очистка в индексах
- вакуум сканирует индексные блоки в физическом порядке: 1,2,3,4,5,6,7,8



Исключение блоков из структуры индекса

- после удаления строк в таблице вакуум может исключать блоки из структуры индекса
  - блоки остаются в файлах индекса
  - помечаются как свободные в карте свободного пространства (fsm) индекса
  - могут использоваться повторно встраиваясь в структуру индекса и в другом месте дерева
- число уровней не уменьшается вакуумом
- "правый" блок каждого уровня индекса не может быть удалён из дерева
- если у корневого и внутренних (промежуточных) блоков один наследник, то он становится "fastroot" - быстрым корнем
  - его адрес и его уровень сохраняется в блоке метаданных в полях fastroot и fastlevel
  - поиск по индексу начинается с fastroot,а не root



Число исключённых блоков из структуры индекса

- запрос для получения числа исключённых из структуры индекса блоков:

```
select type, count(*) from bt_multi_page_stats('t_pkey',1,-1) group by type;
 type | count
------+-------
 D    |     6
 r    |     1
 l    |   552
 i    |     4
 d    |  2181
```

- type: d - удалёный листовой (deleted leaf) блок, D - удалённый внутренний (deleted internal) блок
- второй столбец в запросе показывает число блоков
- изменения LP_DEAD не передаются на реплику, на реплике биты LP_DEAD при сканировании по индексу не выставляются
- при вакуумировании читаются все блоки в файлах индекса
  - чем меньше блоков в файлах индекса, тем быстрее будет работать автовакуум

Практика

- Часть 1. Методы доступа
- Часть 2. Использование индексов ограничениями целостности
- Часть 3. Характеристики btree индексов
- Часть 4. Навигация по структуре btree индексов
- Часть 5. Дедупликация в btree индексах
- Часть 6. Индексы в убывающем порядке
- Часть 7. Покрывающие индексы и Index Only Scan
- Часть 8. Частичные (partial) индексы
- Часть 9. Изучение структуры индекса типа btree
- Часть 10. Очистка блоков индекса при его сканировании
- Часть 11. Медленное выполнение запросов на реплике из-за отсутствия очистки блоков индекса
- Часть 12. Определение числа удаленных строк
- Часть 13. Поиск по структуре индекса btree

8-1

TOAST

TOAST (The Oversized-Attribute Storage Technique)

- TOAST это техника хранения "атрибутов" (полей) большого размера
- поля большого размера - это поля не помещающиеся в блок
- позволяет хранить поля размером до 1Гб
- если в таблице есть заведомо большое поле или вставляется строка с большим полем, то создается toast-таблица и индекс на toast-таблицу
- в TOAST выносятся отдельные поля строк
- вынесенные поля делятся на части (chunk) по 1996 байт:

```
postgres@tantor:~$ pg_controldata | grep TOAST
Maximum size of a TOAST chunk:        1996
postgres@tantor:~$ /usr/lib/postgresql/15/bin/pg_controldata
-D /var/lib/postgresql/15/main| grep TOAST
Maximum size of a TOAST chunk:        1988
postgres@vanilla-x32:~$ pg_controldata | grep TOAST
Maximum size of a TOAST chunk:        2000
```

Поля переменной длинны

- строка должна поместиться в один блок
- поля varlena, которые не помещаются в блок выносятся в таблицу TOAST
- типы данных фиксированной ширины не сжимаются и не выносятся в TOAST
- стратегию хранения можно установить командой `ALTER TABLE имя ALTER COLUMN имя SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN | DEFAULT }`.
- сжатие поддерживается для MAIN и EXTENDED
- **1 байт в начале поля переменной ширины хранит длину поля длиной до 127 байт**
- **4 байта в начале поля переменной ширины хранит длину поля для полей длиннее 126 байт**

Вытеснение полей в TOAST

- в СУБД Tantor и PostgreSQL выносятся поля строк длинной больше 2032 байт
  - при этом поля режутся на части по 1996 байт
- алгоритм (очередность) вытеснения зависит от порядка следования столбцов
- при доступе к каждому вытесненному полю дополнительно читается 2-3 блока TOAST-индекса

```
select reltoastrelid, reltoastrelid::regclass
from pg_class where relname='t';
 reltoastrelid |      reltoastrelid
---------------+-------------------------
         74295 | pg_toast.pg_toast_74292
\d+ pg_toast.pg_toast_74292
TOAST table "pg_toast.pg_toast_74292"
   Column   |  Type   | Storage
------------+---------+---------
 chunk_id   | oid     | plain
 chunk_seq  | integer | plain
 chunk_data | bytea   | plain
Owning table: "public.t"
Indexes:
    "pg_toast_74292_index" PRIMARY KEY,
            btree (chunk_id, chunk_seq)
Access method: heap
select chunk_id, chunk_seq, length(chunk_data)
from pg_toast.pg_toast_74292;
 chunk_id | chunk_seq | length
----------+-----------+--------
    74297 |         0 |   1996
    74297 |         1 |      9
```

Алгоритм вытеснения полей в TOAST

- при вставке строки в таблицу она полностью размещается в памяти серверного процесса в строковом буфере размера 1Гб (или 2Гб
- при обновлении строки обработка выполняется по затрагиваемым командой полям в пределах строкового буфера. Поля, не затрагиваемые командой, представлены в буфере заголовком длинной 18 байт.
- после обработки (сжатия или выноса) каждого поля проверяется размер строки. Если размер не превышает `toast_tuple_target` (по умолчанию 2032 байт) строка сохраняется в буфер и обработка строки заканчивается
- обработка начинается с поля EXTENDED и EXTERNAL от наибольшего размера к меньшему
- сжатие и вынос полей MAIN выполняется только после выноса всех полей EXTENDED и EXTERNAL

Toast chunk

- при использовании EXTERNAL поля размером от 1997 байт создают второй чанк небольшого размера из-за которого в блок TOAST помещается только 3 чанка большого размера
- для полей EXTERNAL с размером от 1997 до ~2300 байт есть вероятность менее плотного хранения

```
select reltoastrelid, reltoastrelid::regclass
from pg_class where relname='t';
 reltoastrelid |      reltoastrelid
---------------+-------------------------
         74295 | pg_toast.pg_toast_74292
\d+ pg_toast.pg_toast_74292
TOAST table "pg_toast.pg_toast_74292"
   Column   |  Type   | Storage
------------+---------+---------
 chunk_id   | oid     | plain
 chunk_seq  | integer | plain
 chunk_data | bytea   | plain
Owning table: "public.t"
Indexes:
    "pg_toast_74292_index" PRIMARY KEY,
            btree (chunk_id, chunk_seq)
Access method: heap
select chunk_id, chunk_seq, length(chunk_data)
from pg_toast.pg_toast_74292;
 chunk_id | chunk_seq | length
----------+-----------+--------
    74297 |         0 |   1996
    74297 |         1 |      9
```

Ограничения TOAST

- в любой таблице в TOAST может быть вынесено не больше 2^32 полей (~4млрд.)
- для вынесенного в TOAST поля в блоке таблицы хранится 18 байт, они не выравниваются, но вся строка выравнивается
- значение, остающееся в блоке таблицы, после выноса поля в TOAST:

```
select lp_len, t_data from heap_page_items(get_raw_page('t', 0));
 lp_len |                    t_data
--------+-------------------------------------------
     42 | \x0112d9070000d5070000392201003722 0100
```

Выравнивание строк с вытесненными в TOAST полями

- выравниваются как отдельные поля, так и вся строка
- каждая строка выравнивается по 8 байт

Пример:

Две таблицы:

```
create table t3 (c1 text);
create table t4 (c1 serial , c2 smallint, c3 text);
```

вставить строки с полем большой длинны, так чтобы все поля были вытеснены в TOAST:

```
DO $$ BEGIN
FOR i IN 1 .. 200 LOOP
insert into t3 VALUES (repeat('a',1000000)); insert into t4 VALUES (default, 1, repeat('a',1000000));
END LOOP; END; $$ LANGUAGE plpgsql;
```

то в блок обоих таблиц помещается одинаковое число строк:

```
select count(*) from heap_page_items(get_raw_page('t3','main',0)) where (t_ctid::text::point)[0]=0 union
select count(*) from heap_page_items(get_raw_page('t4','main',0)) where (t_ctid::text::point)[0]=0;
 count
-------
   156
(1 row)
```

Место, занимаемое таблицами тоже одинаковое

- в таблице t4 можно сохранить два столбца за счет места, которое в первой таблице не могло использоваться

Параметры `toast_tuple_target` и `default_toast_compression`

- на вытеснение влияют два макроса: TOAST_TUPLE_THRESHOLD и TOAST_TUPLE_TARGET
  - по умолчанию равны **2032**
- если размер строки больше TOAST_TUPLE_THRESHOLD, то начинается сжатие и/или вытеснение полей строки
- поля будут сжиматься и рассматриваться на предмет хранения в TOAST, пока оставшиеся поля не поместятся в TOAST_TUPLE_TARGET
- TOAST_TUPLE_TARGET можно переопределить только **на уровне таблицы**:
- `ALTER TABLE t SET (toast_tuple_target = 2032);`
- **TOAST_TUPLE_THRESHOLD не переопределяется**
- параметр конфигурации `default_toast_compression` устанавливает алгоритм сжатия:
- `ALTER SYSTEM SET default_toast_compression = pglz;`

Оптимизация Heap Only Tuple

- при обновлении (`UPDATE`) строки в индексы могут не вноситься изменения
- изменения не выходят за пределы блока таблицы (heap only)
- условия:
  - изменяются только поля, не входящие ни в один из индексов (кроме индексов типа brin) на таблицу
  - новая версия строки размещается в том же блоке, что и прежняя версия
- если новая версия строки размещается в блоке, отличном от того в котором находится прежняя версия строки, то:
  - прежняя версия станет последней в цепочке HOT-версий
  - во всех индексах на таблице будут созданы новые записи, указывающие на новую версию строки

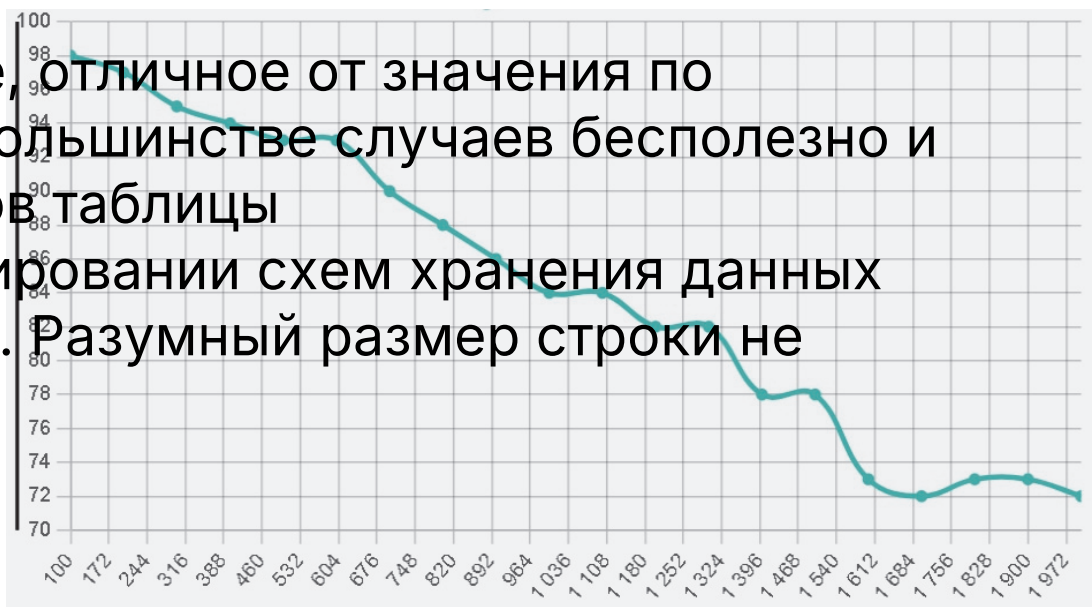
Мониторинг HOT update

- статистика HOT доступна в представлениях `pg_stat_all_tables` и `pg_stat_user_tables`
- **счетчик HOT обновлений собирается по каждой таблице и отражается в столбце: `n_tup_hot_upd`**
- все обновления отражаются в столбце `n_tup_upd`
- **случаи, если при обновлении не нашлось места для новой версии строки и цепочка HOT была оборвана, а новая версия была вставлена в другой блок показывает `n_tup_newpage_upd`**
- статистика по базе обнуляется вызовом функции `pg_stat_reset();`
  - после вызова функции рекомендуется выполнить ANALYZE по всей базе

```
 select relname, n_tup_upd, n_tup_hot_upd, n_tup_newpage_upd,
round(n_tup_hot_upd*100/n_tup_upd,2) as hot_ratio from pg_stat_all_tables where n_tup_upd<>0
order by 5;
     relname     | n_tup_upd | n_tup_hot_upd | n_tup_newpage_upd | hot_ratio
 ----------------+-----------+---------------+-------------------+-----------
  pg_rewrite     |        14 |             9 |                 5 |     64.00
  pg_proc        |        33 |            23 |                10 |     69.00
  pg_class       |     71645 |         63148 |              8351 |     88.00
```

Влияние FILLFACTOR на HOT cleanup

- чтобы HOT cleanup выполнился, предыдущий UPDATE строки должен превысить границу `min(90%, FILLFACTOR)`
- если размер строк в таблице такой, что в блок помещается меньше 9 строк, то восьмая строка не превысит границу 90%, а девятая строка будет больше 11% размера блока и она не поместится в блок
- установка FILLFACTOR в значение, отличное от значения по умолчанию (FILLFACTOR=100) в большинстве случаев бесполезно и только увеличивает размер файлов таблицы
- наиболее эффективно при проектировании схем хранения данных делать размер строк небольшими. Разумный размер строки не больше 600 байт



Внутристраничная очистка в таблицах

- выполняя SELECT и UPDATE, серверный процесс может удалить dead tuples (версии строк, вышедшие за горизонт видимости базы данных, xmin horizon), выполнив реорганизацию версий строк внутри блока
- внутристраничная очистка совместима с HOT и может предварительно освобождать место, которое будет использоваться новыми версиями строк, появляющимися в результате UPDATE
- выполняется, если:
  - блок заполнен более чем на 90% или FILLFACTOR (по умолчанию 100%)
  - ранее выполнявшийся UPDATE не смог разместить новую версию строки в этом блоке
- приблизительная оценка по версиям строк, которые могут быть очищены: `pg_stat_all_tables.n_dead_tup`

Внутристраничная очистка в индексах (повторение)

- выполняется при индексном сканировании
- если строка в таблице удалена, индексная запись на строку или цепочку версий может быть помечена флагом LP_DEAD
- пометка может ставиться командой SELECT
- журнальная запись не создается, но блок грязнится
- помеченная индексная запись игнорируется на мастере, но не на реплике
- помеченная индексная запись будет очищена при выполнении команд, которые меняют данные в таблице

```
create table t (id int primary key, c text) with (autovacuum_enabled = off);
insert into t SELECT i, 'simple delete ' || i from generate_series(1, 1000000) as i;
delete from t where id between 100 and 900000;
analyze t;
explain (analyze, buffers, costs off) select * from t where id between 1 and 900000;
 Index Scan using t_pkey on t (actual time=0.010..218.477 rows=99 loops=1)
   Buffers: shared hit=11489
 Execution Time: 218.600 ms
```

8-2

Типы данных

Типы данных наименьшего размера: `boolean`, `"char"`, `char`, `smallint`

- список типов данных и их характеристики есть в таблице `pg_type`
- если столбец будет использоваться для поиска, стоит оценить эффективность индексирования столбцов, составных индексов, эффективность сканирования индекса доступными способами (Bitmap Index Scan, Index Scan, Index Only Scan)
- Типы данных, занимающие наименьшее место:
  - `boolean` занимает 1 байт
  - `"char"` занимает 1 байт, хранит символы ASCII
  - `char` занимает 2 байта,
    - хранит символы в кодировке базы данных
  - `smallint`, занимает 2 байта
    - хранит целые числа от -32768 до 32767

Типы данных переменной длинны

- для строк переменной длинны стоит использовать тип text
- размерность для text не указывается
- занимаемое место:
  - один байт, если длинна поля меньше 127 байт и строка пустая ''
  - если кодировка UTF8, то ASCII символы занимают 1 байт. Поэтому значение '1' займет 2 байта: \x0531. Значение '11' займёт 3 байта: \x073131. поле состоящее из буквы 'э' займет 3 байта: \x07d18d
  - если длинна поля больше 126 символов, то заголовок поля станет 4 байта и поля будут выравниваться по 4 байта
- поля могут сжиматься и оставаться в блоке
- поля могут выноситься в TOAST, оставляя при этом в блоке 18 байт (не выравниваются)
- двоичные данные стоит хранить в типе данных bytea. Это тип данных переменной длинны и его поведение такое же, как у типа text

Целочисленные типы данных

- целые числа можно хранить в типах `int(integer, int4), bigint (int8), smallint (int2)`
- обычно используются для столбцов PRIMARY KEY
- `bigint` выравнивается по 8 байт
- `int` для первичного или уникального ключа ограничит число строк в таблице 4млрд (2^32)
- для генерации значений для типов `smallint, int` и `bigint` используются последовательности и есть синонимы `smallserial(serial2), serial (serial4), bigserial (serial8)`
- для хранения чисел может использоваться тип переменной длинны `numeric` (синоним `decimal`), **накладные расходы 4 байта на хранение длинны поля**

Выбор типов данных для первичного ключа

- при меньшем числе строк размер индекса по столбцу типа uuid существенно больше, чем по столбцу типа bigint
- дополнительные функции устанавливается расширением:

```
create extension uuid-ossp;
ERROR:  syntax error at or near "-"
create extension "uuid-ossp";
CREATE EXTENSION
```

- размер индекса по столбцу первичного ключа и пример теста:

```
psql -c "create table tt1 (id bigint generated by default as identity (cache 60) primary key, data bigint);"
pgbench -T 30 -c 4 -f txn.sql 2> /dev/null | grep tps
tps = 2693
psql -c "select count(*), pg_indexes_size('tt1'), pg_table_size('tt1') from tt1;"
 count | pg_indexes_size | pg_table_size
-------+-----------------+---------------
 80760 |         1884160 |       3612672
psql -c "create table tt1 (id uuid default gen_random_uuid() primary key, data bigint);"
pgbench -T 30 -c 4 -f txn.sql 2> /dev/null | grep tps
tps = 2338
psql -c "select count(*), pg_indexes_size('tt1'), pg_table_size('tt1') from tt1;"
 count | pg_indexes_size | pg_table_size
-------+-----------------+---------------
 70115 |         2777088 |       3760128
```

Параметр cache у последовательностей

- параметр последовательностей и identity столбцов
- по умолчанию `cache=1`

```
create table tt1 (id bigint generated by default as identity (cache 1) primary key)
```

- не стоит увеличивать значение
- выборка значений из последовательности не является узким местом
- установка в большее значение приводит к неэффективным вставкам в индекс, созданным на столбец в который вставляются сгенерированные значения
- увеличение значения приводит к замедлению вставок в индекс

Хранение дат, времени, их интервалов

- для хранения дат, времени, интервалов используются типы:
  - date (4 байта, с точностью до дня)
  - `timestamp, timestamptz, time` точность до микросекунды, размер одинаковый 8 байт, содержимое одинаковое
  - `timetz` - длинна 12 байт, `interval` - длинна 16 байт
- типы данных `timestamp, timestamptz` не хранят часовой пояс
- `timestamptz` приводит хранимое время к временной зоне клиента
- `timestamptz` физически хранит значения в UTC

```
create table t(t TIMESTAMP, ttz TIMESTAMPTZ);
insert into t values (CURRENT_TIMESTAMP, CURRENT_TIMESTAMP);
set timezone='UTC';
select t, ttz from t;
 2024-11-25 23:19:47.833968 | 2024-11-25 20:19:47.833968+00
update t set ttz=t;
select lp_off, lp_len, t_hoff, t_data from heap_page_items(get_raw_page('t','main',0)) order by lp_off;
 lp_off | lp_len | t_hoff |                t_data
--------+--------+--------+-----------------------------------
   8096 |     40 |     24 | \x70580939c1ca020070580939c1ca0200
   8136 |     40 |     24 | \x7044c4bcc3ca020070580939c1ca0200
select t, ttz from t;
 2024-11-25 20:19:47.833968 | 2024-11-25 20:19:47.833968+00
```

Функции проверки типа данных и размера поля

- для проверки типа значения используется функция `pg_typeof(значение)`
- функция получения размера поля и полного размера строки `pg_column_size(строка или поле)`
- при хранении строки в блоке ее размер должен быть кратен 8 байт, если он меньше, то будет добавлено от 1 до 7 байт
- для датавременных типов есть тип date (4 байта, с точностью до дня), остальные типы `timestamp, timestamptz, time` имеют точность до микросекунды, размер 8 байт. Типы `timetz` и `interval` имеют длину 12 и 16 байт и из-за длинны не стоит их использовать.

```
pg_typeof(now()) -> timestamp with time zone
pg_typeof(now()::date) -> date
pg_typeof(current_date) -> date
pg_typeof(1.1) -> numeric
pg_column_size(1.1) -> 10
pg_column_size(1.1::float) -> 8
интервальный тип: pg_column_size(interval '1s') -> 16
размер строки из трёх полей  в удачном порядке: pg_column_size(row(true::boolean, 1::int4, 1::int8)) -> 40
неудачная перестановка столбцов: pg_column_size(row(true::boolean, 1::int8, 1::int4)) -> 44
размер строки с ещё одним порядком: select pg_column_size(row(1::int4, 1::int8, true::boolean))  -> 41;
```

Типы данных для вещественных чисел

- фиксированной ширины, с плавающей точкой, с округлением до 6 или 15 разрядов (значащих чисел в десятичном формате):
  - `float4 , real , float(1..24)` - в 4 байтах хранит не менее 6 разрядов
  - `float8 , float , double precision , float(25..53)` в 8 байтах хранит не менее 15 разрядов

```
select 12345678901234567890123456789.1234567890123456789::float4::numeric;
       12345700000000000000000000000
select 123456789012345678901234567890.1234567890123456789::float8::numeric;
       123456789012346000000000000000
```

- переменной ширины, **без потери точности** при вычислениях:
  - `numeric , decimal`
  - точность можно задать параметрами: `numeric(presision,scale)`

```
select 1234567890123456789.123456789::numeric + 0.000000000000000000000123456789::numeric;
       1234567890123456789.123456789000000000000123456789
```

- пример экспоненциальной записи одинаковых чисел с разной мантиссой и порядком:

```
select 12345.6::float4, '12.3456e+03'::float4, '123.456e+02'::float4,'1234.56e+01'::float4;
1.235e+04 | 1.235e+04 | 1.235e+04 | 1.235e+04
```

Параметр конфигурации `extra_float_digits`

- параметром `extra_float_digits` можно уменьшить число цифр в текстовом представлении чисел float8, float4 и геометрических типов
- диапазон значений от -15 до 3 включительно
- значения 1,2,3 эквивалентны
- параметр влияет только на отображение, на вычисления и приведения к типу numeric не влияет

```
show extra_float_digits;
 1
select 1234567890.123456789::float8, 1.123456789::float4;
       1234567890.1234567     |      1.1234568
set extra_float_digits = 0;
select 1234567890.123456789::float8, 1.123456789::float4;
       1234567890.12346       |      1.12346
set extra_float_digits = -5;
select 1234567890.123456789::float8, 1.123456789::float4;
       1234567890             |      1
reset extra_float_digits;
select 234567890.199999989::float8::numeric, 1.19999999123::float4::numeric;
       234567890.2            |              1.2
```

Хранение вещественных чисел

- обычно в столбцах хранятся хранятся числа небольшой размерности
- тип numeric для небольших целых чисел хранит данные компактнее, чем float8 (точность 15 цифр после точки)
- типы float8 и float4 теряют точность целочисленных разрядов, numeric не теряет
- точности float4 может быть недостаточно: 6 десятичных разрядов
- точность float8 15 десятичных разрядов

```
create table t5( c1 double precision, c2 real, c3 numeric);
insert into t5 values (1,1,1),(1.0/3, 1.0/3, 1.0/3),(1111,1111,1111),(1111.11,1111.11, 1111.11);
select lp_off, lp_len, t_hoff, t_data from heap_page_items(get_raw_page('t5','main',0)) order by lp_off;
lp_off |lp_len |t_hoff |                    t_data
-------+-------+-------+-----------------------------------------------
 7984 |    43 |    24 |\x3d0ad7a3705c914085e38a440f008157044c04
 8032 |    41 |    24 |\x00000000005c914000e08a440b00805704
 8080 |    49 |    24 |\x555555555555d53fabaaaa3e1b7f8a050d050d050d050d050d
 8136 |    36 |    24 |\x000000000000f03f0000803f0b00800100
```

Разрядность результата деления numeric

```
 * For inherently inexact calculations such as division and square root,
 * we try to get at least this many significant digits; the idea is to
 * deliver a result no worse than float8 would.
 */
#define NUMERIC_MIN_SIG_DIGITS          16
```

- не менее 16 значащих цифр, то есть не хуже, чем float8
- не меньше, чем разрядность любого из входных параметров

```
insert into t5 values (1,1, 1.00000000000000000000000000000000001/3);
select lp, lp_off, lp_len, t_hoff, t_data from heap_page_items(get_raw_page('t5','main',0)) order by
lp desc limit 1;
 lp | lp_off | lp_len | t_hoff |                                                                  t_data
----+--------+--------+--------+--------------------------------------------------------------------------------
  5 |   7912 |     57 |     24 | \x000000000000f03f0000803f2b7f92050d050d050d050d050d050d050d060d
(5 rows)
select * from t5 order by ctid desc limit 1;
 c1 | c2 |                  c3
----+----+--------------------------------------
  1 |  1 | 0.33333333333333333333333333333334
```

- Посмотреть описание операторов можно командой psql:

```
\doS+ /                                       List of operators
   Schema   | Name |  Left arg type |  Right arg type |   Result type |   Function   | Description
------------+------+----------------+-----------------+---------------+--------------+-------
 pg_catalog | /    | bigint         | bigint          | bigint        | int8div      | divide
 pg_catalog | /    | numeric        | numeric         | numeric       | numeric_div  | divide
...
```

Практика

- Часть 1. TOAST
- Часть 2. Структура таблиц TOAST
- Часть 3. Эффективность UPDATE в сравнении с INSERT
- Часть 4. HOT cleanup
- Часть 5. Мониторинг HOT cleanup
- Часть 6. Типы данных небольшого размера
- Часть 7. Хранение типов данных переменной длинны
- Часть 8. Тип данных для столбца первичного ключа
- Часть 9. Типы данных для хранения дат и времени
- Часть 10. Типы данных float и real

9-1

Архитектура

Запуск экземпляра, процесс postgres

- запускается процесс postgres
- считываются файлы параметров и комбинируются с параметрами, переданными в командной строке
- проверяются разрешения на директорию PGDATA
- проверяется наличие управляющего файла `pg_control`
- выделяется память, загружаются разделяемые библиотеки, регистрируются обработчики событий
- в PGDATA создается файл postmaster.pid наличие которого и правильность номера процесса проверяется раз в минуту
- процесс postgres регистрирует серверные сокеты с параметрами TCP_NODELAY и TCP_KEEPALIVE, создается файл UNIX-сокета
- читается файл `pg_hba.conf`
- запускается процесс startup и фоновые процессы

Процесс startup

- процесс startup определяет состояние кластера по управляющему файлу. Возможные состояния:
  - "`shut down`" после корректной (с контрольной точкой) остановки экземпляра
  - "`shut down in recovery`" экземпляр реплики сбойнул, когда находился в режиме восстановления
  - "`in production`" экземпляр был остановлен без контрольной точки
  - "`shutting down`" экземпляр сбойнул в процессе остановки
  - "`in crash recovery`" экземпляр мастера сбойнул при восстановлении
  - "`in archive recovery`" экземпляр сбойнул в процессе управляемого восстановления
  - "`unrecognized status code`" состояние неизвестно
- процесс startup выполняет синхронизацию файлов PGDATA и их восстановление по файлам WAL

```
postgres@tantor:~$ pg_controldata | grep state
Database cluster state:               shut down
```

Синхронизация PGDATA, параметр `recovery_init_sync_method`

- процесс startup выполняет выполняет синхронизацию PGDATA способом, указанным в параметре конфигурации `recovery_init_sync_method`
- по умолчанию `recovery_init_sync_method=`**`fsync`**, что означает, что процесс startup будет открывать и посылать fsync по ВСЕМ файлам в PGDATA:

```
LOG:  database system was interrupted; last known up at 03:17:41 MSK
LOG:  syncing data directory (fsync), elapsed time: 10.00 s, current path: ./base/5/4066825
LOG:  syncing data directory (fsync), elapsed time: 20.00 s, current path: ./base/5/1903193
...
LOG:  syncing data directory (fsync), elapsed time: 70.00 s, current path: ./base/5/4081550
LOG:  database system was not properly shut down; automatic recovery in progress
```

- можно установить значение параметра конфигурации `recovery_init_sync_method=`**`syncfs`**. При этом значении выполняется sync в целом на смонтированных файловых системах, на которых расположены PGDATA

Синхронизация бэкапа, параметр `pg_basebackup --sync-method`

- в 17 версии у утилиты `pg_basebackup` появился параметр `--sync-method` с двумя значениями fsync и syncfs
  - значение по умолчанию fsync
- до 17 версии можно использовать параметр `--no-sync` (`-N`) и затем команду операционной системы `sync -f`.
- при большом числе файлов в кластере, а значит и в создаваемых бэкапах использование syncfs или sync позволяют уменьшить время на резервирование с той же отказоустойчивостью. Пример:

```
rm -rf /var/lib/postgresql/backup/1
time pg_basebackup -c fast -D $HOME/backup/1 -P
3783151/3783151 kB (100%), 1/1 tablespace
real    2m50.010s
user    0m52.035s
sys     0m3.515s
rm -rf /var/lib/postgresql/backup/1
time pg_basebackup --sync-method=syncfs -c fast -D $HOME/backup/1 -P
real    0m40.807s
user    0m36.967s
sys     0m2.612s
```

Параметр `restart_after_crash`

- по умолчанию значение `on` и это повышает доступность экземпляра, так как при отключении экземпляр будет просто принудительно остановлен
- определяет будут ли перезапущены процессы экземпляра при сбое серверного процесса:

```
postgres=# select name, setting, context, max_val, min_val from pg_settings where name ~ 'restart';
        name         | setting | context | max_val | min_val
---------------------+---------+---------+---------+---------
 restart_after_crash | on      | sighup  |         |
(1 row)
```

- при изменении значения достаточно перечитать конфигурацию
- параметр может быть установлен в значение `off`, если экземпляр управляется кластерным программным обеспечением и оно решает запустить экземпляр или нет

Особенности работы экземпляра в контейнере docker

- изменяемые файлы, в частности PGDATA, должны лежать на томах (volumes)
- работа в контейнере не добавляет высокой доступности
- процесс postgres **не** должен иметь PID=1
- при создании и запуске контейнера нужно использовать параметр `docker run -d --init`

```
root@tantor:~# docker exec имя_контейнера ps
  PID USER      TIME COMMAND
    1 postgres  0:38 postgres
   31 postgres  0:09 postgres: logger
   32 postgres  0:45 postgres: checkpointer
   33 postgres  0:38 postgres: background writer
...
root@tantor:~# docker rm -f имя_контейнера
root@tantor:~# docker run --init -d -e POSTGRES_USER=postgres -e POSTGRES_PASSWORD=postgres -e
POSTGRES_INITDB_ARGS="--data-checksums" -e POSTGRES_HOST_AUTH_METHOD=trust -p 5434:5434 -e
PGDATA=/var/lib/postgresql/data -d -v /root/data:/var/lib/postgresql/data --name имя_контейнера postgres
```

Что происходит при запуске серверного процесса

- для каждой сессии создается серверный процесс
- получает структуру (часть памяти) PGPROC из списка свободных и устанавливает поля в начальные значения
- списки свободных PGPROC для серверных процессов хранятся в поле FreeProcs структуры PROC_HDR, все они находятся в разделяемой памяти
- Инициализируются (выделяются и заполняются) три кэша в локальной памяти серверного процесса:
  - Кэш для быстрого доступа к таблицам (RelationCache)
  - Кэш таблиц системного каталога (CatalogCache)
  - Кэш планов выполнения команд (PlanCache)
- выделяется память под менеджер "порталов" TopPortalContext
- клиент аутентифицируется
- догружаются разделяемые библиотеки, указанные в параметрах `session_preload_libraries` и `local_preload_libraries`
- выделяется память MessageContext для команд и процесс готов к приему команд

Общая память процессов экземпляра

- может существовать больше 72 структур
- размеры большинства структур можно посмотреть в представлении `pg_shmem_allocations`
- размер некоторых структур меняется параметрами конфигурации
- размеры всех хэш-структур выдаются не верно, как 2944

```
 select * from (select *, lead(off) over(order by off) - off as true_size from
pg_shmem_allocations) as a order by 1;
                 name         |    off    |   size    | allocated_size | true_size
------------------------------+-----------+-----------+----------------+-----------
 <anonymous>                  |           |   4946048 |        4946048 |
 Archiver Data                | 147726208 |         8 |            128 |       128
...
 XLOG Recovery Ctl            |   4377728 |       104 |            128 |       128
                              | 148145024 |   2849920 |        2849920 |
(60 rows)
 select * from (select *, lead(off) over(order by off) - off as true_size from
pg_shmem_allocations) as a where a.true<>a.allocated_size order by 1;
 LOCK hash                    | 142583808 | 2896      |           2944 |    695168
 PREDICATELOCK hash           | 144423680 | 2896      |           2944 |   1260416
...
```

Кэш таблиц системного каталога

- выделяется в локальной памяти каждого процесса в контексте CacheMemoryContext
- при создании или удалении объекта, процесс зафиксировавший транзакцию, посылает сообщение в кольцевой буфер shmInvalBuffer разделяемой памяти
- буфер хранит до 4096 сообщений
- процессы потребляют сообщения и обновляют свои локальные кэши
- если процесс пропустит сообщения, то полностью очистит свой локальный кэш таблиц системного каталога (CatalogCache) и будет его заново заполнять

```
 select * from (select *, lead(off) over(order by off) - off as true_size
from pg_shmem_allocations) as a where name='shmInvalBuffer' order by 1;
       name       |    off    | size  | allocated_size | true_size
------------------+-----------+-------+----------------+-----------
 shmInvalBuffer | 146865024 | 68128 |          68224 |     68224
(1 row)
```

Представление pg_stat_slru

- в PGDATA есть поддиректории, в которых сохраняются служебные данные кластера
- для ускорения доступа на чтение-запись в файлы этих директорий используются кэши в разделяемой памяти экземпляра
- статистика используется для установки параметров конфигурации, задающих размеры SLRU-кэшей

```
select name, blks_hit, blks_read, blks_written, blks_exists, flushes, truncates from pg_stat_slru;
       name        | blks_hit | blks_read | blks_written | blks_exists | flushes | truncates
-------------------+----------+-----------+--------------+-------------+---------+-----------
 commit_timestamp  |        0 |         0 |            0 |           0 |     103 |         0
 multixact_member  |        0 |         0 |            0 |           0 |     103 |         0
 multixact_offset  |        0 |         3 |            2 |           0 |     103 |         0
 notify            |        0 |         0 |            0 |           0 |       0 |         0
 serializable      |        0 |         0 |            0 |           0 |       0 |         0
 subtransaction    |        0 |         0 |           26 |           0 |     103 |       102
 transaction       |   349634 |         4 |           87 |           0 |     103 |         0
 other             |        0 |         0 |            0 |           0 |       0 |         0
(8 rows)
```

Локальная память процесса

- доступна только одному процессу, поэтому блокировки для доступа к ней не нужны
- большая часть структур не занимает много памяти и интересны только для понимания алгоритмов работы процессов
- параметры, наиболее сильно влияющие на выделение локальной памяти процесса:
  - `work_mem` - выделяется для обслуживание узлов (шагов) плана выполнения (если шаги способны выполняться одновременно), в том числе каждым параллельным процессом. Вместе с параметром `hash_mem_multiplier` влияет на память, выделяемую каждым серверным и параллельным процессом.
  - `maintenance_work_mem` значение по умолчанию 64MB. Задает объем памяти, выделяемый каждым процессом (серверным, параллельным), участвующем в выполнении команд `VACUUM`, `ANALYZE`, `CREATE INDEX`, `ALTER TABLE ADD FOREIGN KEY`

Представление pg_backend_memory_contexts

- показывает память, выделенную серверным процессом, обслуживающим **текущую** сессию
- контекст памяти (memory contexts) - набор частей памяти (chunks), которые выделяются процессом для выполнения какой-то задачи
- для выполнения подзадачи может выделяться дочерний контекст
- Контексты образуют дерево (иерархию)
- в представлении иерархию отображают столбцы: name (название контекста памяти), parent (название родительского контекста памяти, level
- в столбце ident содержится детализация того, что хранится в контексте

```
select sum(total_bytes), sum(used_bytes), sum(free_bytes) from pg_backend_memory_contexts;
   sum    |   sum    |  sum
---------+---------+--------
 2114816 | 1380760 | 734056
```

Функция pg_log_backend_memory_contexts(PID)

- начиная с 17 версии у команды EXPLAIN есть опция memory (по умолчанию отключена), которая выдает сколько памяти использовал планировщик и общую память серверного процесса
- память чужих сессий можно вывести в диагностический лог кластера функцией:

```
postgres=# SELECT pg_log_backend_memory_contexts(111);
 pg_log_backend_memory_contexts
--------------------------------
 t
(1 row)
LOG: statement: SELECT pg_log_backend_memory_contexts(111);
...
LOG: logging memory contexts of PID 111
LOG: level: 0; TopMemoryContext: 60528 total in 5 blocks; 16224 free (6 chunks); 44304 used
LOG: level: 1; TopTransactionContext: 8192 total in 1 blocks; 6728 free (0 chunks); 1464 used
...
LOG: level: 2; AV dblist: 8192 total in 1 blocks; 7840 free (0 chunks); 352 used
LOG: Grand total: 658848 bytes in 38 blocks; 270616 free (32 chunks); 388232 used
```

9-2

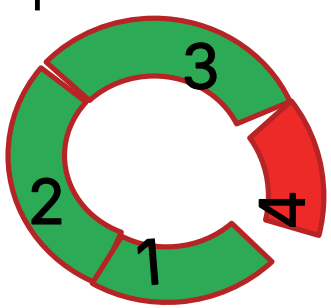
Блокировки

Типы блокировок

- spinlock (циклическая проверка)
  - Используются для очень краткосрочных действий - не дольше нескольких десятков инструкций процессора
  - Средств мониторинга нет
- легковесные (LWLocks)
  - Используются для доступа к структурам в разделяемой памяти
  - Имеют монопольный (на чтение и изменение) и разделяемый режим (чтение)
  - не больше 200 одновременно
- обычные (тяжеловесные)
  - Автоматически освобождаются по окончании транзакции
  - Есть несколько уровней блокировок
  - Обслуживают блокировки **12 типов**, в том числе advisory locks.
- предикатные блокировки (SIReadLock) используются транзакциями с уровнем изоляции SERIALIZABLE

Параметры deadlock_timeout и log_lock_waits

- время ожидания получения блокировки, через которое ожидающий процесс начнет проверку на наличие взаимоблокировок
- в одном цикле ожидания одна проверка
- по умолчанию 1 секунда, значение стоит увеличить хотя бы до длительности выполнения типичной для приложения транзакции
- рекомендуется установить `log_lock_waits = true`, чтобы получать сообщения в диагностический журнал что какой-либо процесс ждет дольше, чем `deadlock_timeout`
- наличие взаимоблокировки это ошибка в архитектуре приложения
- взаимно заблокироваться могут два или более процесса ("кольцо" взаимоблокировки)



Параметр lock_timeout

- по умолчанию не действует (значение ноль)
- не стоит устанавливать на уровне кластера
- действует на явные и неявные попытки получить блокировку

```
set lock_timeout = '1s';
ALTER TABLE test ADD COLUMN ts timestamp DEFAULT clock_timestamp();
ERROR:  canceling statement due to lock timeout
```

- granted=f означает, что блокировка не получена (ShareLock другого процесса препятствует получению):

```
select pid, relation::regclass::text, mode, granted, fastpath from pg_locks
where locktype='relation' and database = (select oid from pg_database where
datname=current_database()) order by pid, fastpath desc;
  pid   | relation |        mode         | granted | fastpath
--------+----------+---------------------+---------+----------
 33210  | test     | AccessExclusiveLock | f       | f
 43344  | test     | ShareLock           | t       | f
```

Подтранзакции

- подтранзакции это точки сохранения
  - используются чтобы откатиться, а не переводить транзакцию в состояние сбоя
- создаются
  - командой SAVEPOINT
  - секцией EXCEPTION в блоке на языке pl/pgsql
- в структуре PGPROC сохраняется до 64 номеров подтранзакций
- подтранзакциям которые только читают данные присваивается виртуальный номер
- если встречается команда изменения данных, то подтранзакциям вплоть до основной транзакции присваиваются реальные номера
- рекомендация: не писать часто исполняемый код, который порождает больше 64 подтранзакций, его выполнение станет узким местом (падение TPS на ~25%)
- В СУБД Tantor есть параметр `subtransaction_buffers`

Мультитранзакции

- блокировка FOR NO KEY UPDATE устанавливается командой UPDATE, которая не вносит изменения в ключевые столбцы
- блокировка FOR KEY SHARE устанавливается командой DELETE и UPDATE которая обновляет значения ключевых столбцов
- также разделяемые блокировки устанавливаются командами SELECT .. FOR SHARE, FOR NO KEY UPDATE, FOR KEY SHARE
- разделяемые блокировки позволяют одновременно работать со строкой нескольким транзакциям
- одновременная работа реализуется "мультитранзакциями", которые имеют свой счетчик `xid`, свои файлы и кэши
- соответствие между `xid` транзакций и мультитранзакций хранятся в директории `PGDATA/pg_multixact`
- advisory locks не являются заменой блокировок строк, так как их количество ограничено

Быстрый путь блокирования

- предназначен для уменьшения накладных расходов на получение и освобождение блокировок на relations (секции таблиц, обычные таблицы, индексы), которые часто запрашиваются, но редко конфликтуют
- используются для "слабых" блокировок: AccessShare, RowSharek, RowExclusive которые устанавливают команды SELECT, INSERT, UPDATE, DELETE, MERGE
- наличие "сильной" (Share, ShareRowExclusive, Exclusive, AccessExclusive) блокировки на relation, попадающий в 1/1024 часть таблицы блокировок, не дает использовать быстрый путь, поэтому:
  - Если установить сильную блокировку на таблицу, то команды, работающие с ней, не смогут использовать быстрый путь
  - не стоит выполнять большое число команд, устанавливающих сильные блокировки во время большой нагрузки на экземпляр

Сильные и слабые блокировки таблиц

- слабые блокировки могут быть получены по быстрому пути
- сильные блокировки таблиц препятствуют установку слабых блокировок по быстрому пути
- автовакуум и автоанализ не мешают использовать быстрый путь
- автовакуум и автоанализ не блокируют команды, в случае конфликта блокировок рабочий процесс автовакуума прерывает обработку таблицы

	ACCESS SHARE	ROW SHARE	ROW EXCL.	SHARE UPDATE EXCL.	**SHARE**	**SHARE ROW EXCL.**	**EXCL.**	**ACCESS EXCL.**
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCLUSIVE					X	X	X	X
SHARE UPDATE EXCL.				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCLUSIVE			X	X	X	X	X	X
EXCLUSIVE		X	X	X	X	X	X	X
ACCESS EXCLUSIVE	X	X	X	X	X	X	X	X

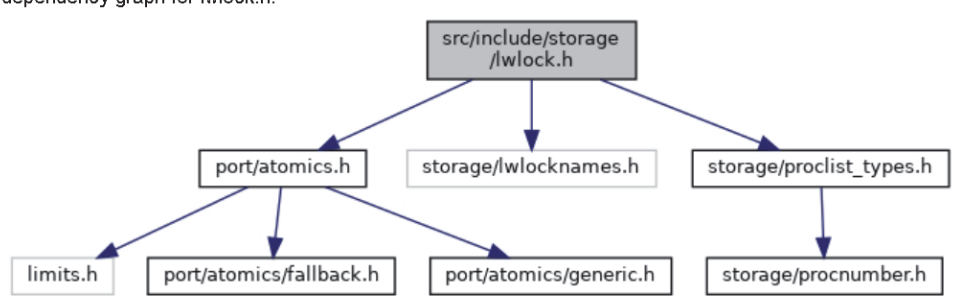
Секции таблицы блокировок

- таблица блокировок разделена на 16 разделов (partitions, секций, частей)
- в PGPROC каждого процесса сохраняется список частей, которые он блокирует
- число легковесных блокировок, которые используются для защиты структур памяти таких как lock partition, ограничено 200 (`MAX_SIMUL_LWLOCKS`)

lwlock.h File Reference

```
#include "port/atomics.h"
#include "storage/lwlocknames.h"
#include "storage/proclist_types.h"
```

Include dependency graph for lwlock.h:

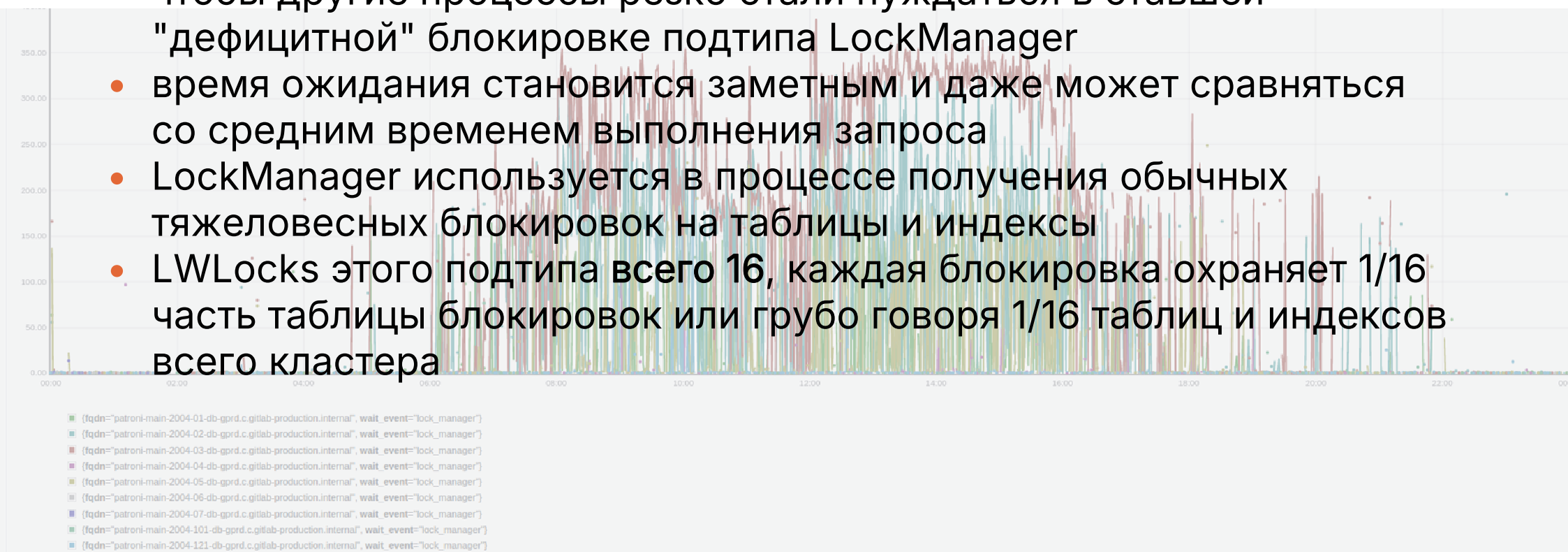


Транши блокировок (tranches)

- число легковесных блокировок 200 это общее ограничение, количество блокировок для каждого подтипа LWLock (которых больше 73) имеет меньшее ограничение и называется траншем
- для работы с таблицей тяжеловесных блокировок одновременно можно использовать 16 траншей (NUM_LOCK_PARTITIONS=16)
  - по обычному пути только 16 процессов одновременно может находиться в процессе получения блокировки на объект
- для работы с описателем буферов 128 траншей (`NUM_BUFFER_PARTITIONS=128`)
- названия легковесных блокировок - значение столбца `wait_event` для `wait_event_type='LWLock'` представления `pg_stat_activity`

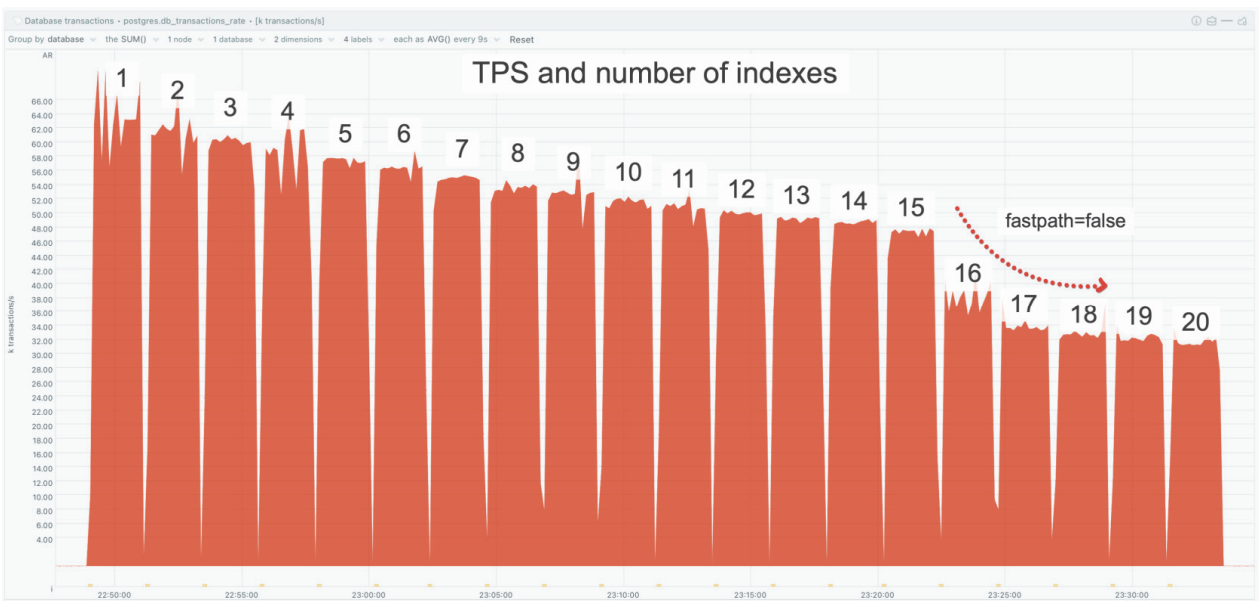
Легковесные блокировки

- LWLocks должны удерживаться микросекунды
- задержка в сотни или тысячи микросекунд - этого достаточно, чтобы другие процессы резко стали нуждаться в ставшей "дефицитной" блокировке подтипа LockManager
- время ожидания становится заметным и даже может сравняться со средним временем выполнения запроса
- LockManager используется в процессе получения обычных тяжеловесных блокировок на таблицы и индексы
- LWLocks этого подтипа **всего 16**, каждая блокировка охраняет 1/16 часть таблицы блокировок или грубо говоря 1/16 таблиц и индексов всего кластера



Блокирование по быстрому пути и 16 блокировок

- первые 16 блокировок типа AccessShareLock, RowShareLock, RowExclusiveLock процесс получает по быстрому пути
- для остальных используется более медленный метод
- при большом числе активных процессов и большом числе `fastpath=false` блокировки `wait_event='LockManager'` в `pg_stat_activity` выходят в топ и становятся узким местом

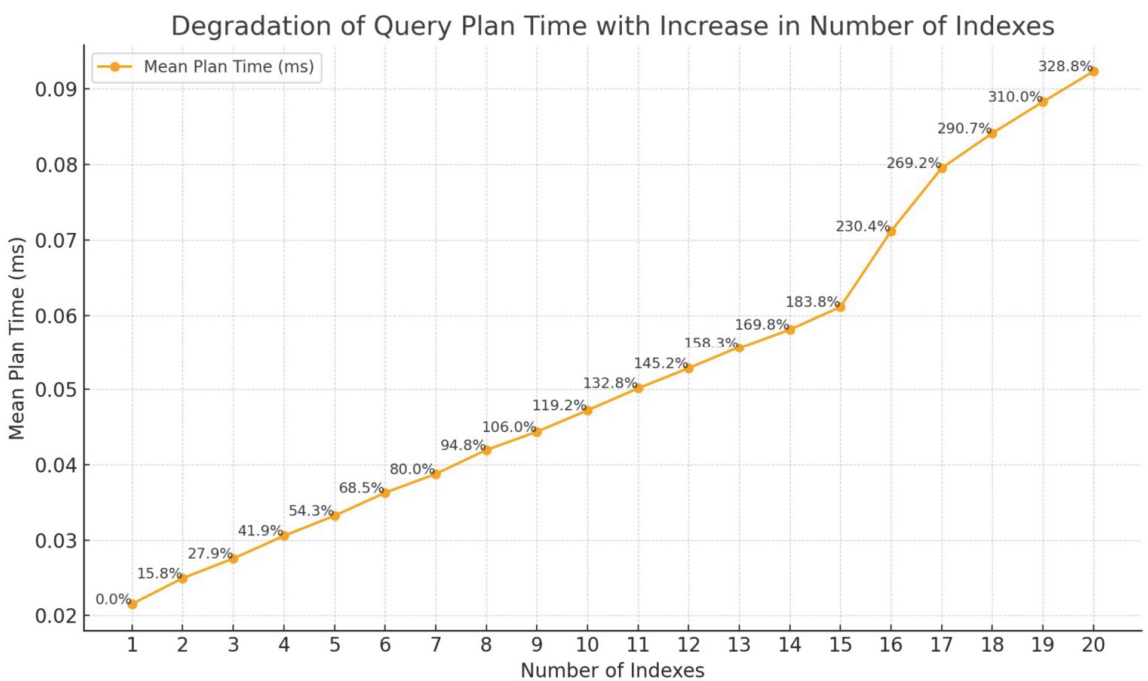


Индексы, соединения, секции и быстрый путь

- на этапе планирования на все таблицы, индексы, секции таблиц, которые рассматриваются планировщиком устанавливаются блокировки типа AccessShareLock
- после создания плана блокировки с тех relations, которые **отсутствуют в плане** снимаются

Рекомендации:

- использовать подготовленные запросы
- использовать меньше индексов
- использовать меньше соединений
- использовать меньше секций



Параметр `join_collapse_limit`

- при соединении больше 8 таблиц эффективность снижается
- `from_collapse_limit` (по умолчанию 8) задаёт максимальное число "элементов" в списке FROM, **до** которого планировщик будет объединять вложенные запросы с внешним запросом
- `join_collapse_limit` (по умолчанию равен `from_collapse_limit`) задаёт максимальное число "элементов" с фразой JOIN, **до** достижения которого планировщик будет полностью перебирать последовательность соединений
- `geqo_threshold` (по умолчанию 12) минимальное число "элементов", при котором для планирования запроса станет использоваться генетический алгоритм

```
drop table if exists t cascade;
create table t (id numeric, c text);
insert into t values (1, 'a');
create or replace view t1 as select * from t;
select * from t1 as ta join (select * from t) as tb on (ta.id=tb.id) join
generate_series(1, 3) as tc(id) on (tb.id=tc.id);
```

Представление `pg_locks`

- выдает все тяжеловесные блокировки на экземпляре (по всем базам данных)
- Число тяжеловесных блокировок ограничено на экземпляре произведением `max_locks_per_transaction * (max_connections + max_prepared_transactions)`
- `max_prepared_transactions` относится не к подготовленным командам, а распределенным транзакциям, по умолчанию значение ноль
- `max_locks_per_transaction` по умолчанию 64
- `max_connections` по умолчанию 100
- в столбце `locktype` представления `pg_locks` указан один из 12 типов блокировок: **`relation`**, `extend`, `frozenid`, `page`, `tuple`, `transactionid`, `virtualxid`, `spectoken`, `object`, `userlock`, **`advisory`**, `applytransaction`

Параметр `track_commit_timestamp`

- значение по умолчанию false и включать не стоит, так как экземпляр эти данные не использует
- в директории `PGDATA/pg_commit_ts` будет сохраняться метка времени фиксации транзакции
- данные о времени фиксации транзакции могут использоваться расширением pglogical для процедур разрешения конфликтов
- временные метки можно получить функциями:
  - `pg_last_committed_xact()`
  - `pg_xact_commit_timestamp()`
  - `pg_xact_commit_timestamp_origin()`
- значение параметра сохраняется в управляющем файле
- изменение значения требует рестарт экземпляра

Практика

- Часть 1. Запуск экземпляра в контейнере docker
- Часть 2. Разделяемая память экземпляра
- Часть 3. Локальная память серверного процесса
- Часть 4. Логирование памяти процесса в диагностический журнал
- Часть 5. Взаимоблокировки при проведении тестов
- Часть 6. Мультитранзакции
- Часть 7. Пример теста

10

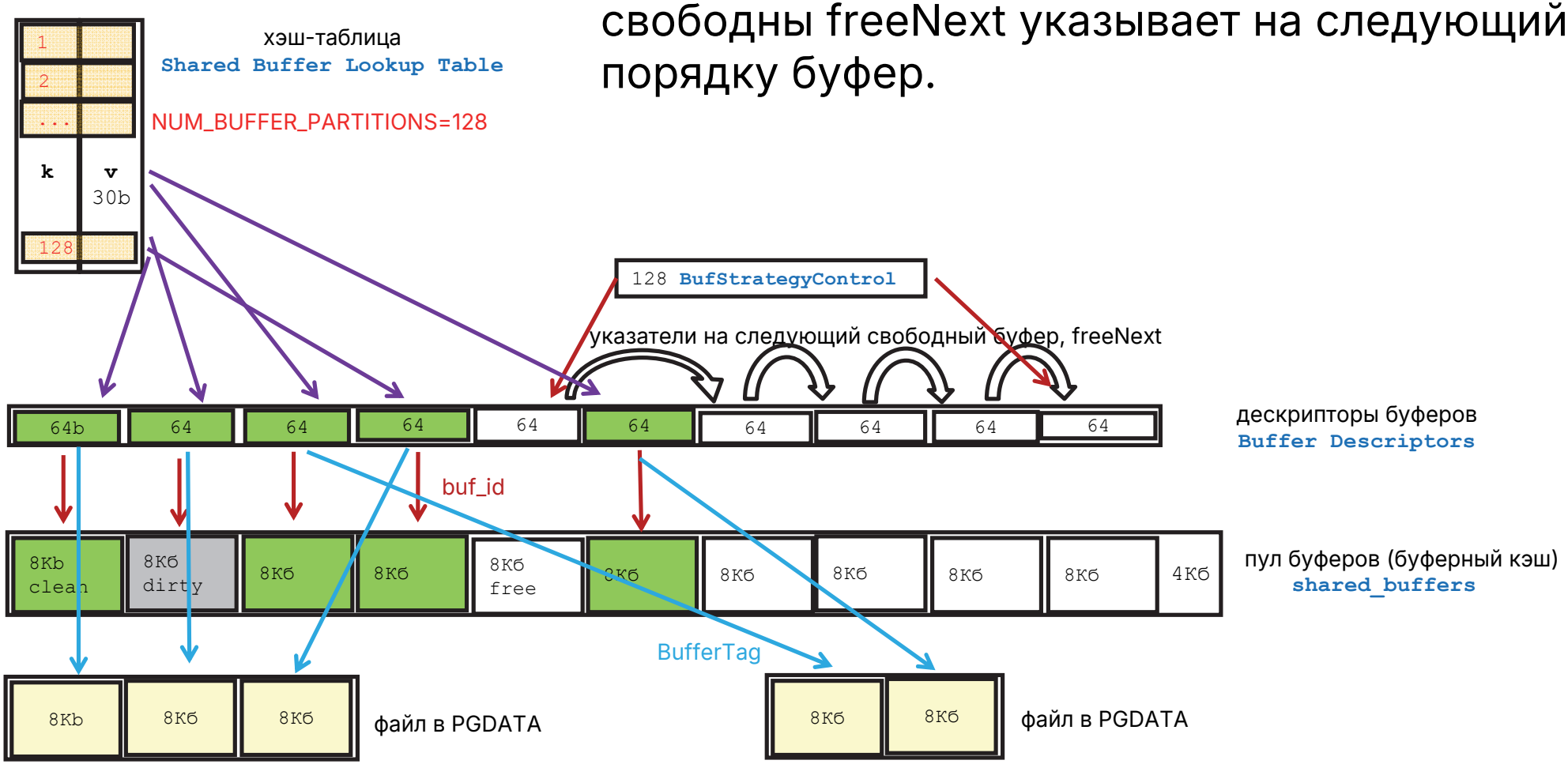
Буферный кэш

Структуры памяти, обслуживающие буферный кэш

- **`Buffer Blocks`** - сам буферный кэш
  - выделяется память по числу буферов*8192 байта плюс 4096 байт
  - параметр конфигурации `shared_buffers` задаёт число буферов
- **`Buffer Descriptors`** - описатели (заголовки) буферов
  - выделяется память по числу буферов * размер описателя (64 байта)
- в каждом описателе хранится:
  - адрес блока на диске в виде метки блока (BufferTag)
    - адрес блока содержит: ТБС, БД, файл, форк, номер блока от начала первого файла
  - адрес буфера в виде порядкового номера буфера в буферном кэше
- описатель связан 1:1 (один к одному) с буфером
- данных в 20 байтах которые занимает BufferTag достаточно (не нужно никуда обращаться), чтобы считать блок с диска

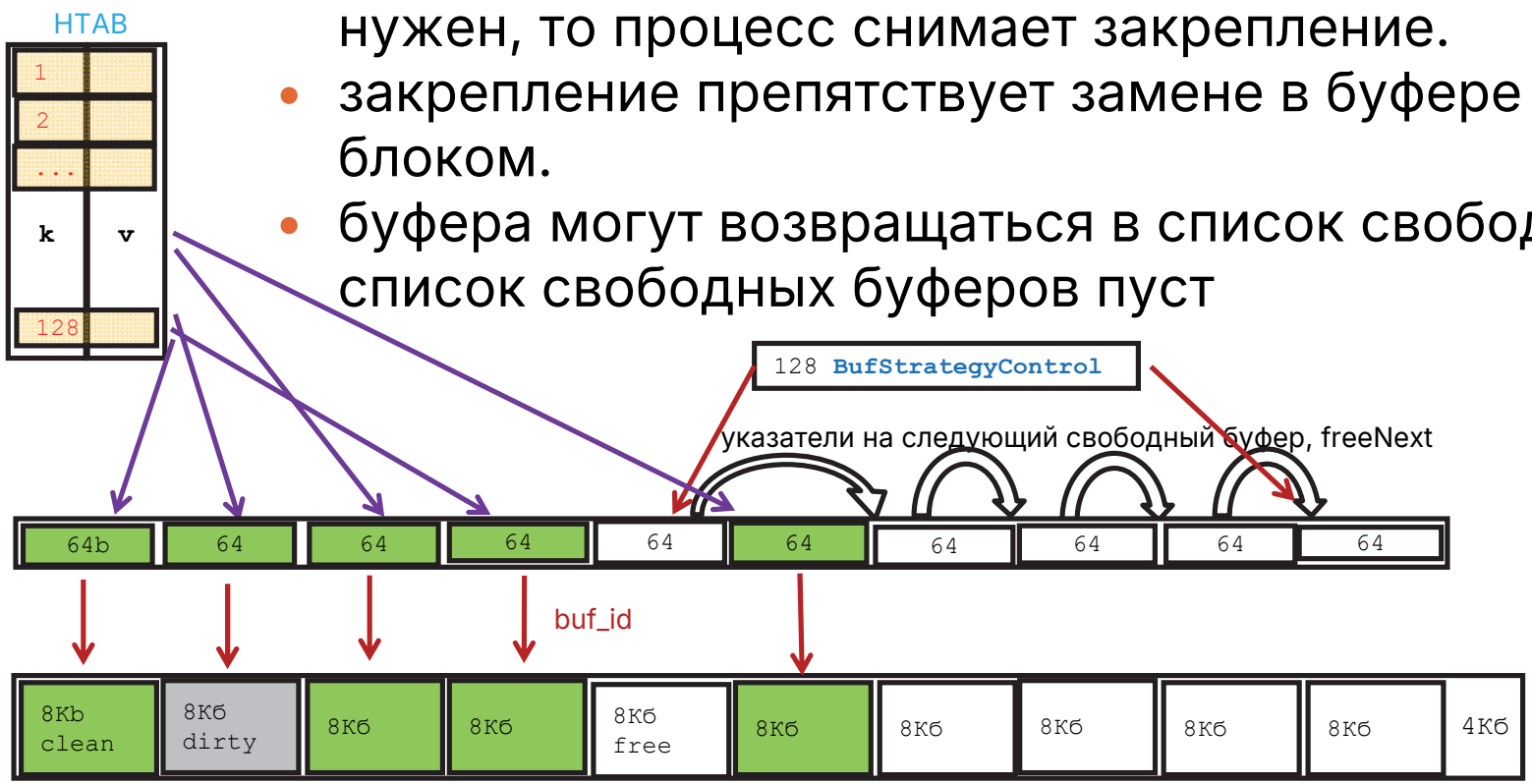
Структуры памяти, обслуживающие буферный кэш

- после запуска экземпляра, пока все буфера свободны freeNext указывает на следующий по порядку буфер.



Поиск свободного буфера

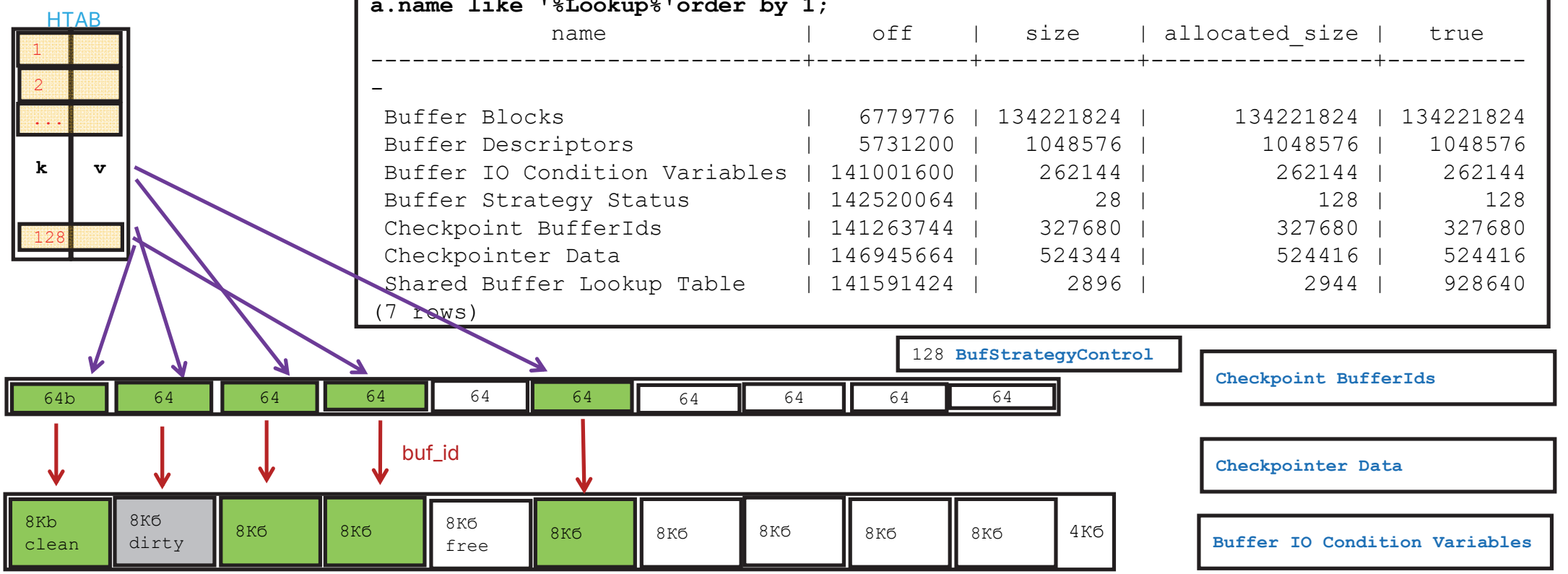
- если процесс хочет работать с блоком, то он его ищет в буферном кэше. Если находит, то закрепляет. Множество процессов может закрепить буфер. Если процессу буфер не нужен, то процесс снимает закрепление.
- закрепление препятствует замене в буфере блока другим блоком.
- буфера могут возвращаться в список свободных. Обычно список свободных буферов пуст



Алгоритм вытеснения грязных буферов

- процесс checkpointer перед посылкой грязных блоков на запись сортирует их отдельно по каждому файлу

```
select * from (select *, lead(off) over(order by off) - off as true from
pg_shmem_allocations) as a where a.name like 'Buf%' or a.name like 'Check%' or
a.name like '%Lookup%'order by 1;
              name                |    off    |   size    | allocated_size |   true
----------------------------------+-----------+-----------+----------------+----------
-
 Buffer Blocks                    |   6779776 | 134221824 |      134221824 | 134221824
 Buffer Descriptors               |   5731200 |   1048576 |        1048576 |   1048576
 Buffer IO Condition Variables    | 141001600 |    262144 |         262144 |    262144
 Buffer Strategy Status           | 142520064 |        28 |            128 |       128
 Checkpoint BufferIds             | 141263744 |    327680 |         327680 |    327680
 Checkpointer Data                | 146945664 |    524344 |         524416 |    524416
 Shared Buffer Lookup Table       | 141591424 |      2896 |           2944 |    928640
(7 rows)
```



Стратегии замены буферов

- число записей в BufTable это сумма NBuffers и NUM_BUFFER_PARTITIONS
- методы замены блоков в буферном кольце (buffer cache replacement strategy):
  - BULKREAD. Для последовательного чтения блоков таблиц (Seq Scan) размер которых **не меньше 1/4 кэша** буферов используется набор буферов в буферном кэше размером 256Кб
  - VACUUM. Грязные страницы не убираются из кольца, а посылаются на запись. Размер кольца задается параметром конфигурации `vacuum_buffer_usage_limit`. По умолчанию 256Кб.
  - BULKWRITE. Используется командами COPY и CREATE TABLE AS SELECT. Размер кольца 16Мб.
- в буферном кэше блок может находиться только в одном буфере
- если буфер стал грязным, он исключается из буферного кольца.

Поиск блока в буферном кэше

Процесс экземпляра:

- создает в своей локальной памяти экземпляр структуры BufferTag
- вычисляет 4-байтный хэш от BufferTag
- по значению хэша определяет номер партиции в хэш-таблице `Shared Buffer Lookup Table`
- запрашивает легковесную (LWLock) блокировку типа BufMappingLock партиции таблицы в которую попало значение хэша
- в хэш-таблице находит порядковый номер блока в кэше буферов или -1 если блока нет в кэше

Закрепление буфера (pin) и блокировка content_lock

- закрепление (pin) используется для того, чтобы блок в буфере не был заменен на другой
- для чтения или изменения содержимого блока в буфере нужна легковесная блокировка content_lock, ссылка на которую сохраняется в описателе блока `Buffer Descriptors`
- блокировка должна удерживаться короткое время, в отличие от pin
- для удаления места занимаемого строкой (HOT cleanup или vacuum) после pin и Exclusive процесс дожидается, чтобы у других процессов не стало pin (то есть pincount=1)
- для добавления в блок новой строки или изменения xmin, xmax существующих строк процесс должен получить блокировку content_lock типа Exclusive
- если процесс имеет pin и Shared content_lock, то он может менять некоторые биты в t_infomask, в частности статус фиксации/отката

Освобождение буферов при удалении файлов

- при удалении базы данных выполняется полное сканирование всех дескрипторов буферов
- полное сканирование дескрипторов выполняется, если размер удаляемого relation больше 1/32 пула буферов
- в остальных случаях поиск дескрипторов идет через хэш-таблицу
- при увеличении кэша буферов с 1Гб до 16Гб время на поиск буферов при удалении таблицы увеличивается на порядок
- поиск выполняется при удалении файлов в результате DROP, TRUNCATE и вакуума (если не отключено усечение файлов)

```
pgbench --file=CreateAndDrop.sql -j 1 -c 1 -T 10
TPS для HP 128MB 417
TPS для HP  1GB  375
TPS для HP  4GB  227
TPS для HP  8GB  127
TPS для HP 16GB   55
TPS для HP 18GB   33
```

CreateAdndDrop.sql:

```
create table x(id int);
insert into x values (1);
drop table x;
```

Оптимизированное расширение файлов

- для расширения размера файлов используются две функции `mdzeroextend(SMgrRelation reln, ForkNumber forknum, BlockNumber blocknum, int nblocks, bool skipFsync)` и `mdextend(..)`
- функция `mdzeroextend(..)` появилась в 16 версии и может расширять файл сразу несколькими блоками (8Кб) за один вызов к операционной системе
- при расширении файлов более, чем на 8 блоков (64Кб) используется системный вызов `posix_fallocate()`. При этом:
  - команда на забивание блоков нулями на диск не передаётся, но будет считаться, что блоки забиты нулями
  - для добавляемых блоков не выделяется место под страницы (по 4Кб) в страничном кэше linux
  - файловые системы могут не выделять реальное место на диске (delayed allocation) и помечать в метаданных, что диапазоны блоков содержат нулевые значения (sparse file)

Изменение размера файлов и буферный кэш

- если в файле нет места, то файлы relations расширяются вызовами вплоть до 64 блока
- автовакуум и вакуум могут на последнем этапе обработки таблицы усечь последний файл форка main или даже удалить файлы форка, если не меньше чем в **тысяче блоков** с конца форка нет строк
  - Если монопольная блокировка на таблицу не будет получена за **5 секунд**, то усечение не выполняется
  - Если монопольная блокировка будет получена, то **через хэш-таблицу** ищутся буфера с блоками, которые будут усечены
  - Если не будут найдены, то полностью сканируются дескрипторы буферов

Предварительное чтение блоков (prefetch)

- предварительное чтение блоков данных в страничный кэш linux
- реализуются вызовом к операционной системе вызовом `posix_fadvise(VfdCache[file].fd, offset, amount, POSIX_FADV_WILLNEED)`
- для мониторинга нет ли задержек связанных с этим вызовом есть **событие ожидания DataFilePrefetch**
- индексный метод доступа Bitmap Heap Scan использует prefetch на фазе сканирования блоков таблицы
- вакуум и анализ также использует prefetch. Ограничение на количество буферов устанавливается параметром `maintenance_io_concurrency`
- в 17 версии появился параметр `io_combine_limit`
  - По умолчанию 128Кб
  - Диапазон значений от 8Кб до 256Кб
  - Устанавливает сколько блоков можно объединить в один вызов к linux

Представление pg_stat_recovery_prefetch

- предварительное чтение используется процессом startup
- параметр recovery_prefetch
  - установлен по умолчанию в значение try
  - используется процессом startup при запуске экземпляра и на репликах
- параметр wal_decode_buffer_size
  - устанавливает размер журнальных записей, которые процесс startup заранее читает, чтобы определить блоки для предварительной загрузки в кэш буферов
  - значение по умолчанию 512Кб

```
select * from pg_stat_recovery_prefetch \gx
stats_reset     | 2044-01-01 11:11:11.000000+03
prefetch        | 2242   число prefetched блоков
hit             | 52138  не загружались, уже находились в кэше буферов
skip_init       | 139    не загружались, так как инициализированлись нулями
skip_new        | 6      не загружались, так как блоки не существовали
skip_fpw        | 1868   не загружались, так как в WAL был полный образ блоков
skip_rep        | 52272  команда prefetched уже была послана
wal_distance    | 0      количество блоков WAL, прочитанных заранее для prefetch
block_distance  | 0      число блоков в процессе предвыборки
io_depth        | 0      число необработанных вызовов prefetch
```

Расширение pg_prewarm

- стандартное расширение
- устанавливается добавлением библиотеки в параметр:
  `alter system set shared_preload_libraries='pg_prewarm';`
- сохраняет в текстовый файл в корне директории PGDATA адрес каждого блока, который загрузит в кэш буферов после перезапуска экземпляра

```
postgres=# select autoprewarm_dump_now();
 autoprewarm_dump_now
----------------------
                  264
(1 row)
postgres@tantor:~$ head $PGDATA/autoprewarm.blocks
<<264>>
0,1664,1262,0,0
0,1664,6100,0,0
5,1663,1259,0,0
...
```

```
postgres=# \dconfig *prewarm*
    List of configuration parameters
             Parameter              | Value
------------------------------------+-------
 pg_prewarm.autoprewarm             | on
 pg_prewarm.autoprewarm_interval    | 5min
(2 rows)
postgres=# select pg_prewarm('pg_class');
 pg_prewarm
------------
         19
```

Процесс фоновой записи bgwriter

- процесс bgwriter записывает (writeback) грязные буфера и помечает их как чистые
- работа bgwriter снижает вероятность того, что процессы натолкнутся на грязные блоки при поиске буфера-кандидата (victim) на вытеснение (eviction) для замены другим блоком
- "поиск свободного блока" это обычно буфера-кандидата на вытеснение из буфера блока, так как все буфера обычно заняты и список свободных блоков пуст
- при вытеснении грязного блока из буфера обращения к шине ввода-вывода нет, это "writeback": копирование из памяти (буфер) в память (страничный кэш linux)

```
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_flush_after    | 64      | sighup  | 256        | 0
 bgwriter_lru_maxpages   | 100     | sighup  | 1073741823 | 0
 bgwriter_lru_multiplier | 2       | sighup  | 10         | 0
```

Алгоритм очистки кэша буферов процессом bgwriter

- буфер не попадает в список свободных буферов, грязный буфер становится чистым
- на диск записываются только грязные, незакрепленные блоки с `usage_count=0`
- **bgwriter не меняет `usage_count`**
- при включенном подсчете контрольных сумм блок копируется из кэша буферов в локальную память процесса bgwriter и в локальной памяти подсчитывается и сохраняется в заголовке блока контрольная сумма

Представление `pg_stat_bgwriter`

- отражает статистику эффективности работы bgwriter по всему экземпляру
- обнуление статистики: `pg_stat_reset_shared('bgwriter');`
- общее количество записанных на диск буферов: `buffers_backend +buffers_clean +buffers_checkpoint`
- buffers_backend - сколько буферов **очистили** (послали на запись) серверные процессы
- buffers_clean - сколько буферов очистил bgwriter
- **buffers_backend*100/buffers_alloc** в каком проценте случаев серверные процессы сталкивались с грязным буфером, должно быть меньше 1%

```
select * from pg_stat_bgwriter\gx
-[ RECORD 1 ]---------+-------------
checkpoints_timed     | 1567
checkpoints_req       | 97
checkpoint_write_time | 1463587
checkpoint_sync_time  | 59994
buffers_checkpoint    | 86752
buffers_clean         | 4234
maxwritten_clean      | 25
buffers_backend       | 1570184
buffers_backend_fsync | 0
buffers_alloc         | 450550
stats_reset           | 2025-01-01 09:13:
```

Расширение `pg_buffercache`

- стандартное расширение
- создает представление `pg_buffercache`, являющееся обёрткой для функции `pg_buffercache_pages()` и две функции без параметров `pg_buffercache_summary()` и `pg_buffercache_usage_counts()`
- bgwriter не уменьшает `usage_count`, его уменьшают серверные процессы
- распределение буферов по шести "корзинкам" `usage_count` не должно иметь перекоса в одну или другую стороны

```
select * from pg_buffercache_usage_counts();
 usage_count | buffers | dirty | pinned
-------------+---------+-------+--------
           0 |   16000 |     0 |      0
           1 |      17 |     0 |      0
           2 |      23 |     1 |      0
           3 |      43 |     0 |      0
           4 |      24 |     2 |      0
           5 |     277 |     1 |      0
(6 rows)
```

Настройка размера кэша буферов

- размер устанавливается параметром `shared_buffers`
- при изменении значения нужно перезапустить экземпляр
- очистить кэш нельзя, только рестартом экземпляра
- страничный кэш linux можно очистить
- распределение буферов по шести значениям `usage_count` не должно иметь явного перекоса в одну или другую стороны иначе алгоритм вытеснения неэффективен
- для оценки можно использовать столбец `usagecount_avg` значение которого должно быть примерно посередине интервала от 0 до 6 (2.5):

```
select * from pg_buffercache_summary();
buffers_used|buffers_unused|buffers_dirty|buffers_pinned|usagecount_avg
------------+--------------+-------------+--------------+--------------
         384 |        16000 |           0 |            0 | 4.37760416666
```

Параметр `synchronize_seqscans`

- при чтении таблицы размером больше 1/4 буферного кэша табличным методом Seq Scan
  - используется буферное кольцо
  - новый процесс "синхронизируется" с тем процессом, который уже сканирует блоки таблицы для совместного чтения блоков из кольца
  - блоки в буферном кэше, читаются работающими процессами примерно в одно время, а не подгружаются в буферный кэш несколько раз
- порядок строк, возвращаемых запросами, в которых отсутствует предложение ORDER BY, может быть произвольным
- по умолчанию включёно:

```
show synchronize_seqscans;
 synchronize_seqscans
----------------------
 on
```

Практика

- Часть 1. Расширение pg_buffercache
- Часть 2. Буферные кольца
- Часть 3. Расширение pg_prewarm
- Часть 4. Процесс фоновой записи bgwriter

11

Контрольная точка

Контрольная точка

- выполняет процесс checkpointer:
  - периодически по истечении `checkpoint_timeout`
  - по разрастанию журнала `max_wal_size`
  - в конце процедуры остановки или запуска экземпляра
  - продвижении реплики
  - командам checkpoint, create database
  - при резервировании

```
postgres=# checkpoint;
CHECKPOINT
LOG:  checkpoint starting: immediate force wait
LOG:  checkpoint complete: wrote 5 buffers (0.0%);
```

Шаги выполнения контрольной точки

- если экземпляр останавливается, в файл `pg_control` записывается статус о начале гашения экземпляра
- вычисляется LSN следующей журнальной записи. Это будет LSN начала контрольной точки
- ждёт снятия процессами признака DELAY_CHKPT_START
- на диск сбрасываются slru буфера и другие структуры разделяемой памяти
- checkpointer в цикле пробегает все описатели буферов и для грязных блоков устанавливает флаг BM_CHECKPOINT_NEEDED, сохраняет адрес блока (5 чисел) в структуре памяти Checkpoint BufferIds для последующей сортировки
- после установки флага checkpointer не блокирует буфер и блок в буфере может быть сброшен на диск и заменен другим

Шаги выполнения контрольной точки

- сохраненные идентификаторы блоков сортируются в порядке: tblspc, relation, fork, block
- блоки посылаются по одному в страничный кэш linux
- Если checkpoint_flush_after не равен нулю, то выполняется синхронизация по уже отсортированным диапазонам блоков по каждому файлу
- в WAL сохраняется моментальный снимок со списком активных транзакций
- формируется журнальная запись окончания контрольной точки, содержащая LSN журнальной записи, которая была сформирована на момент начала контрольной точки
- в файле `pg_control` сохраняется LSN сформированной журнальной записи
- удаляются WAL-сегменты, которые не должны удерживаться

Параметры конфигурации процесса checkpointer

- `log_checkpoints` по умолчанию `on` начиная с 15 версии
- уменьшать значение `checkpoint_completion_target` не рекомендуется
- `checkpoint_timeout` - основной параметр, влияющий на производительность. Оптимальное значение около 20 минут
- `max_wal_size` второй влияющий на производительность при настройке контрольной точки

```
select name, setting, unit, min_val, max_val, context from pg_settings where name
like '%checkpoint%' or name='max_wal_size';
             name              | setting | unit | min_val |  max_val   | context
-------------------------------+---------+------+---------+------------+---------
 checkpoint_completion_target | 0.9     |      | 0       | 1          | sighup
 checkpoint_flush_after       | 32      | 8kB  | 0       | 256        | sighup
 checkpoint_timeout           | 300     | s    | 30      | 86400      | sighup
 checkpoint_warning           | 30      | s    | 0       | 2147483647 | sighup
 log_checkpoints              | on      |      |         |            | sighup
 max_wal_size                 | 1024    | MB   | 2       | 2147483647 | sighup
(6 rows)
```

Статистика для настройки параметров checkpointer

- серверные процессы при поиске буфера для замены блока на свой не пропускают грязные буфера, они ориентируются только на pin и usagecount
- Кроме checkpointer и серверных процессов сброс грязных блоков выполняет bgwriter
- bgwriter сбрасывает буфера с usagecount=0 и **только у незакрепленных буферов**
- после настройки `checkpoint_timeout` можно настраивать bgwriter: проверять сколько буферов он очистил `pg_stat_bgwriter.buffers_clean` и мог ли очистить:

```
select dirty from pg_buffercache_usage_counts() where pinned=0 and usage_count=0;
 dirty
-------
     0
(1 row)
```

Пример настройки параметров checkpointer

- `select pg_sleep(:t3);` удерживает горизонт базы, что более приближено к реальной нагрузке
- "треугольник" в dirty:

```
select * from pg_buffercache_usage_counts();
 usage_count | buffers | dirty | pinned
-------------+---------+-------+--------
           0 |    6957 |     0 |      0
           1 |      88 |    13 |      0
           2 |      48 |    23 |      0
           3 |      62 |    24 |      0
           4 |      81 |    61 |      0
           5 |    9148 |  7895 |      0
```

```
alter system set checkpoint_timeout = :t3;
select pg_reload_conf();
select pg_current_wal_lsn() AS t1 \gset
select pg_sleep(:t3);
select pg_current_wal_lsn() AS t2 \gset
select pg_size_pretty((:'t2'::pg_lsn - :'t1'::pg_lsn)/:'t3');
```

```
#!/bin/bash
for ((i=300;i<=7200;i+=300))
do
export t3="$i"
psql -f ckpt.sql
done
60  494kB
120 360kB
180 254kB
240 199kB
300 165kB
360 143kB
420 128kB
540 133kB
600  81kB
660  98kB
780  85kB
1200 60kB
1500 53kB
1680 56kB
1800 51kB
2400 45kB
```

Пример настройки параметров checkpointer

- `\! sleep $t3` не удерживает горизонт, TPS увеличивается с ~44 до ~620

```
progress:   600.0 s, 618.6 tps, lat 1.609 ms stddev 1.036
progress:   900.0 s, 629.0 tps, lat 1.582 ms stddev 0.914
progress:  1800.0 s, 627.7 tps, lat 1.585 ms stddev 0.858
progress:  3300.0 s, 615.7 tps, lat 1.616 ms stddev 0.912
progress: 10500.0 s, 618.4 tps, lat 1.609 ms stddev 0.882
progress: 12900.0 s, 618.5 tps, lat 1.609 ms stddev 0.842
```

```
alter system set checkpoint_timeout = :t3;
select pg_reload_conf();
select pg_current_wal_lsn() AS t1 \gset
\! sleep $t3
select pg_current_wal_lsn() AS t2 \gset
select pg_size_pretty((:'t2'::pg_lsn - :'t1'::pg_lsn)/:'t3');
```

```
#!/bin/bash
for ((i=30;i<=600;i+=30))
do
export t3="$i"
psql -f ckpt.sql
done
```

```
30  689kB
60  567kB
90  452kB
120 393kB
150 368kB
180 340kB
210 332kB
240 408kB
270 332kB
300 310kB
330 307kB
360 300kB
390 300kB
420 295kB
450 292kB
480 291kB
```

Практика

- Часть 1. Настройка частоты контрольных точек
- Часть 2. Задержка при запуске экземпляра. Параметр `recovery_init_sync_method`
- Часть 3. Длительность контрольной точки
- Часть 4. Длительность финальной контрольной при остановке экземпляра
- Часть 5. Длительность контрольной точки после падения экземпляра
- Часть 6. Контрольная точка по запросу

12

Автовакуум

Алгоритм вакуумирования

- каждая таблица вакуумируется в отдельной транзакции
- до конца транзакции удерживается моментальный снимок
- если блокировка на таблицу не может быть установлена, то таблица пропускается
- по TOAST-таблицам статистика не собирается
- индексы одной таблицы могут очищаться параллельными рабочими процессами, они могут использовать HugePages для списка TID
- временные объекты очищаются последовательно
- пять фаз вакуумирования каждой таблицы:
  - `SCAN_HEAP`, `VACUUM_INDEX`, `VACUUM_HEAP`, `INDEX_CLEANUP`, `VACUUM TRUNCATE`

Первая фаза вакуумирования

- под накопление TID мертвых строк (LP_DEAD) выделяется память в размере `autovacuum_work_mem`
- в 17 версии если на таблице нет ни одного индекса место в блоках очищается за один проход, а не за два прохода, каждый блок меняется один раз, а не два раза и генерируется меньше журнальных записей
- на первой фазе читаются блоки, в них ставятся биты LP_DEAD и TID всех строк с этим флагом сохраняются в памяти
- до 17 версии память под TID ограничена 1Гб, с 17 версии может использоваться вся `autovacuum_work_mem` и памяти используется в ~20 раз меньше
- если блоков с LP_DEAD будет меньше 2% от всех блоков таблицы и памяти под TID будет использовано меньше 32Мб вакуум заканчивает обработку таблицы и индексы не вакуумируются

Расчёт памяти под TID для вакуумирования

- под накопление TID мертвых строк (LP_DEAD) выделяется память в размере
  - `autovacuum_work_mem` для автовакуума
  - `maintenance_work_mem` для команды VACUUM
  - по умолчанию 64Гб
- память под хранение идентификаторов строк до 17 версии расчитывается по формуле: `maintenance_work_mem = n_dead_tup * 6`. Но не больше 1 гигабайта.
- Пример: 1000000*6/1Мб=5.74 = 6 проходов

```
select schemaname, relname, n_dead_tup, n_live_tup from pg_stat_user_tables where
relname = 'test';
 schemaname | relname | n_dead_tup | n_live_tup
------------+---------+------------+------------
 public     | test    |    1000000 |   10000000
set maintenance_work_mem='1MB';
vacuum verbose test;
INFO:  vacuuming "postgres.public.test"
INFO:  finished vacuuming "postgres.public.test": index scans: 6
```

Вторая и третья фазы вакуумирования

- на второй фазе в индексах очищаются ссылки на TID, сохраненные в памяти на первой фазе
- на третьей фазе в блоках таблиц очищаются TID с LP_DEAD, собранные в памяти на первой фазе: слоты в заголовке блоков становятся пригодными для использования
- замораживаются строки, которые можно заморозить
- обновляются карты свободного места, видимости, заморозки
- для очистки описатель буфера блока блокируется в режиме Exclusive
  - Если получить Exclusive блокировку на описатель буфера не удастся, то блок TID с LP_DEAD не очищается
  - процесс будет пытаться получить блокировку и блок будет очищен, если команда VACUUM использует опции SKIP_LOCKED false или FREEZE или DISABLE_PAGE_SKIPPING, а также если цикл автовакуума запустился в агрессивном режиме (для заморозки строк)

Четвертая и пятая фазы вакуумирования

- фаза усечения файла (файлов) таблицы вызывается если объем пустых блоков в конце последнего файла основного слоя больше 8Мб
  - запрашивается монопольная блокировка на таблицу
  - если блокировка не получена в течение 5 секунд, фаза пропускается
  - фазу можно отключить на уровне таблицы и уровне ее TOAST-таблицы
- анализ выполняется отдельно от вакуума

Агрессивный (Aggressive) режим вакуумирования

- если обычный автовакуум сможет продвигать `pg_class.relfrozenxid` и `pg_class.relminmxid`, то агрессивный автовакуум не понадобится
- агрессивный режим автовакуума запускается, если `age(pg_class.relfrozenxid) > vacuum_freeze_table_age - vacuum_freeze_min_age` **или** `mxid_age(pg_class.relminmxid) > vacuum_multixact_freeze_table_age - vacuum_freeze_min_age`
- `age(pg_database.datfrozenxid) > autovacuum_freeze_max_age` **или** `mxid_age(pg_database.datminmxid) > autovacuum_multixact_freeze_max_age`

```
\dconfig *freeze*
               Parameter                |    Value
----------------------------------------+-------------
 autovacuum_freeze_max_age              | 10000000000
 autovacuum_multixact_freeze_max_age    | 20000000000
 vacuum_freeze_min_age                  |    50000000
 vacuum_freeze_table_age                |   150000000
 vacuum_multixact_freeze_min_age        |     5000000
 vacuum_multixact_freeze_table_age      |   150000000
```

```
SELECT datname, age(datfrozenxid),
mxid_age(datminmxid) FROM
pg_database;
   datname   |    age    | mxid_age
-------------+-----------+---------
 postgres    | 121346629 |       19
 template1   | 121346629 |       19
 template0   | 121346629 |       19
```

Заморозка строк (FREEZE)

- можно и вакуумировать и замораживать TOAST отдельно:

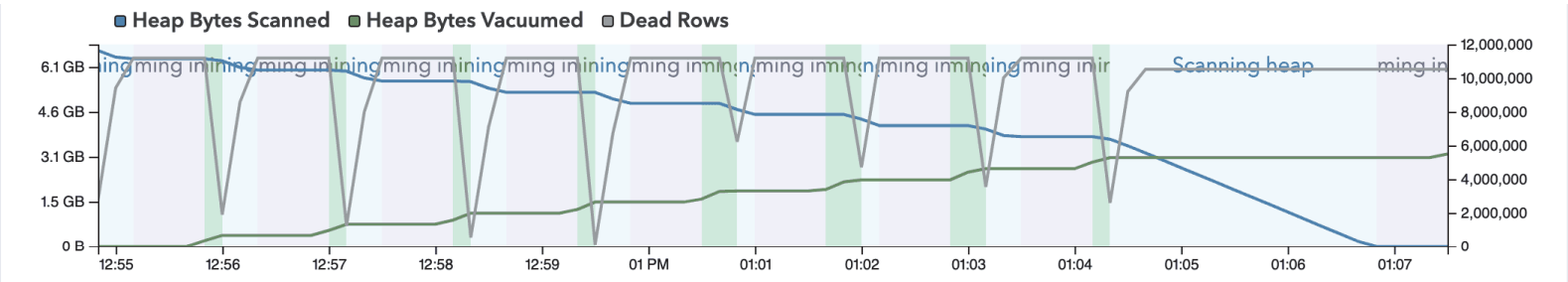
```
 select relname, relfrozenxid, age(relfrozenxid), relminmxid, mxid_age(relminmxid)
from pg_class  where relfrozenxid<>0 order by 3 desc;

vacuum (freeze, verbose) pg_toast.pg_toast_1262;
INFO:  finished vacuuming "postgres.pg_toast.pg_toast_1262": index scans: 0
tuples: 0 removed, 0 remain, 0 are dead but not yet removable, oldest xmin: 105637454
removable cutoff: 105637454, which was 3 XIDs old when operation ended
new relfrozenxid: 105637454, which is 105636725 XIDs ahead of previous value
new relminmxid: 37, which is 36 MXIDs ahead of previous value
frozen: 0 pages from table (100.00% of total) had 0 tuples frozen
```

- Обычный вакуум по возможности (если обработает все блоки которые не обрабатывал раньше и в сумме окажутся замороженными все строки во всех блоках) выполняет заморозку:

```
relname | relfrozenxid | relminmxid
--------+--------------+-----------
 t3     |        67551 |         37
vacuum (verbose) t3;
 relname | relfrozenxid | relminmxid
--------+--------------+-----------
 t3     |    105738379 |         37
```

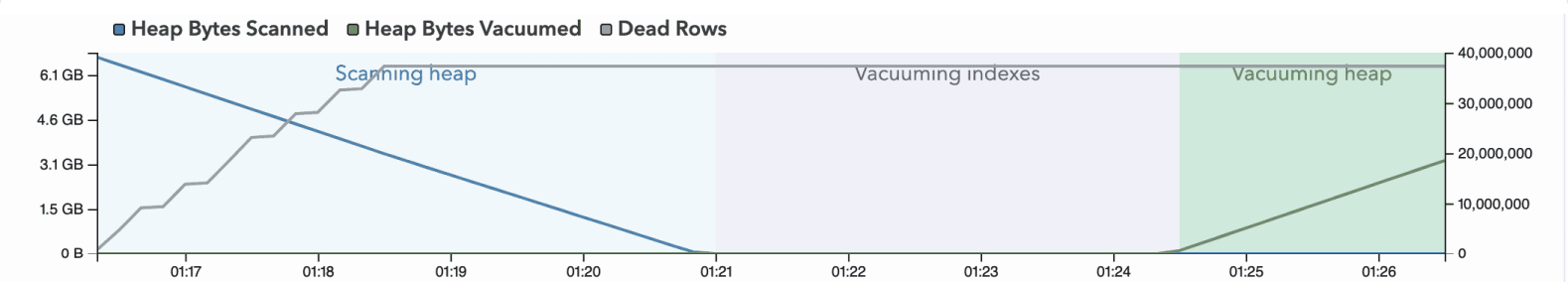
Вакуум в 17 версии PostgreSQL

- использует меньше памяти



Summary Statistics

Time Elapsed	12:53	CPU Time	user: 78.21 s, system: 30.06 s
Index Vacuum Phases	9	Aggressive Vacuum	No



Summary Statistics

Time Elapsed	10:19	CPU Time	user: 78.84 s, system: 47.42 s
Index Vacuum Phases	1	Aggressive Vacuum	No

Сравнительное тестирование вакуума 16 и 17 версий PostgreSQL

- использует меньше памяти в ~20 раз
- память на создание radix tree ограничивается `autovacuum_work_mem` или `maintenance_work_mem`
- использование WAL одинаково в обеих версиях.
- использование процессора на построение и сканирование radix tree (вместо массива) и длительность вакуумирования в 17 версии больше на ~20%
- повышения нагрузки на ввод-вывод нет, только на процессор
  - на avg read/write rate в статистике вакуума влияет CPU usage
- при включенных `(data_checksums=on OR wal_log_hints=on) AND (full_page_writes=on)` объем журнальных записей увеличивается по причине того, что если блок меняется, то один раз после контрольной точки в журнал записывается образ этого блока (неиспользуемое место не записывается)
  - чем реже контрольные точки, тем меньше вероятность многократной записи полного образа блока

Контрольные суммы и WAL

- трудоемкость расчета и проверки контрольных сумм несущественна
- включение контрольных сумм "снижает" производительность тем, что:
  - если выражение `(data_checksums=on OR wal_log_hints=on) AND (full_page_writes=on)` истинно, то при любом изменении содержимого блока между контрольными точками блок запишется в WAL полностью (8Кб минус незанятое место), но только один раз
  - при включенных контрольных суммах блок из буфера копируется в локальную память процесса и оттуда в страничный кэш linux; при копируется из кэша буферов и расходов на копирование блока в локальную память нет
- по умолчанию `wal_log_hints=off`, `full_page_writes=on` и менять значения не стоит
- производительность хоста "снижает" сама работа экземпляра, но это не значит что экземпляр нужно останавливать

Параметры команды VACUUM

- `DISABLE_PAGE_SKIPPING` обрабатывает все блоки таблиц без исключения
- `SKIP_LOCKED false` - не дает пропускать заблокированные объекты, секции таблиц, блоки
- `INDEX_CLEANUP auto/on/off` указывает нужно ли обрабатывать индексы. OFF используется если нужно быстрее убрать мертвые строки из блоков таблиц
- `PROCESS_TOAST false` - отключает обработку таблиц TOAST
- `TRUNCATE false` - отключает пятую фазу
- `PARALLEL` *n*. Число *n* ограничивает число фоновых процессов
- `FULL` полная очистка, использует монопольные блокировки, последовательно устанавливаемые на каждую обрабатываемую таблицу

Параметры команды VACUUM

- `SKIP_DATABASE_STATS` отключает обновление `pg_database.datfrozenxid`
  - позволяет не выполнять полное сканирование таблицы `pg_class`
- `VERBOSE` - выводит статистику выполнения команды
- `FREEZE` - выполняет заморозку строк во всех блоках, кроме тех в которых все строки актуальны и заморожены
- `BUFFER_USAGE_LIMIT` размер буферного кольца вместо `vacuum_buffer_usage_limit`
  - в отличие от параметра конфигурации, `BUFFER_USAGE_LIMIT` можно установить в значение ноль и буферное кольцо не будет использоваться

Расширение `pg_visibility`

- стандартное расширение
- для каждого блока таблицы в карте видимости (слой `_vm`) хранится два бита:
  - `all_visible` все строки в блоке актуальны и видны всем транзакциям
  - `all_frozen` все строки в блоке заморожены
- примеры функций:

```
select * from pg_visibility_map_summary('pg_class');
 all_visible | all_frozen
-------------+------------
          14 |         12
select * from pg_visibility('pg_class',0);
 all_visible | all_frozen | pd_all_visible
-------------+------------+----------------
 f           | f          | f
select * from pg_visibility_map('pg_class');
 blkno | all_visible | all_frozen
-------+-------------+------------
     0 | f           | f
     1 | f           | f
...
```

Мониторинг автовакуума

- на то, что автовакуум с текущими настройками не справляется могут указать:
  - таблицы с мертвыми строками, существенно превышающими порог срабатывания автовакуума;
  - долгая очистка каких-то таблиц;
  - предупреждения в логе
- для мониторинга используются **pg_stat_progress_vacuum, pg_stat_activity, pg_stat_all_tables, pg_class**

```
relation |dead(%)|reltuples|n_dead_tup| effective_settings|  last_vacuumed   | status |..
---------+-------+---------+----------+-------------------+------------------+--------+--
pgbench_ | 19400 |       1 |      194 | vt: 50, vsf: 0.2, | 2024-10-07 12:0 | queued |
branches |       |         |          | DISABLED          | 2:33 (auto)      |        |
pgbench_ |  1160 |      10 |      116 | vt: 50, vsf: 0.2, | 2024-10-07 12:0 | queued |
tellers  |       |         |          | DISABLED          | 2:33 (auto)      |        |

pid | duration | waiting | mode | database | table | phase | table_size | total_size | scanned | vacuumed
----+----------+---------+------+----------+-------+-------+------------+------------+---------+--------
123  01:00:00.0        f  regular  postgres  test   vacuuming indexes 1 GB       2 GB    500 MB   400 MB
```

Представление pg_stat_progress_vacuum

- содержит по одной строке для каждого серверного процесса, выполняющего команду VACUUM и каждого autovacuum worker выполняющих вакуумирование в момент обращения к представлению
- в столбце phase отражается текущая фаза вакуума: `initializing` (подготовительная, проходит быстро), `scanning heap`, `vacuuming indexes`, `vacuuming heap`, `cleaning up indexes`, `truncating heap`, `performing final cleanup` (финальная)
- по столбцам `heap_blks_total` , `heap_blks_scanned` , `heap_blks_vacuumed` оценить ход выполнения очистки
- `num_dead_tuples` - число TID, которые сейчас помещены в структуру памяти. Если достигнет `max_dead_tuples` увеличится значение в **`index_vacuum_count`**
- `VACUUM` **`FULL`** отслеживается через `pg_stat_progress_`**`cluster`**
- `ANALYZE` отслеживается через `pg_stat_progress_analyze`

Параметр `log_autovacuum_min_duration`

- по умолчанию 10 минут
- если автовакуум превысит это время при обработке (вакуум или анализ) таблицы, то в лог кластера запишется сообщение
- при возникновении таких сообщений стоит выяснять причину долгого вакуумирования таблицы

```
LOG:  automatic vacuum of table "postgres.public.pgbench_tellers": index scans: 0
pages: 0 removed, 9 remain, 9 scanned (100.00% of total)
tuples: 125 removed, 10 remain, 0 are dead but not yet removable, oldest xmin: 87738094
removable cutoff: 87738094, which was 2 XIDs old when operation ended
new relfrozenxid: 87738066, which is 609 XIDs ahead of previous value
frozen: 0 pages from table (0.00% of total) had 0 tuples frozen
index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 42 hits, 0 misses, 0 dirtied
WAL usage: 4 records, 0 full page images, 722 bytes
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
LOG:  automatic analyze of table "postgres.public.pgbench_tellers"
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 52 hits, 0 misses, 0 dirtied
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
```

Параметры конфигурации автовакуума

- Список параметров:

```
select name, setting||coalesce(unit,'') unit, context, min_val, max_val, short_desc from
pg_settings where category='Autovacuum'  or name like '%autovacuum%' order by 2 desc;
                 name                 |    unit    |  context  |min_val| max_val
--------------------------------------+------------+-----------+-------+----------------
autovacuum                            | on         | sighup    |       |
autovacuum_naptime                    | 60s        | sighup    | 1     | 2147483
log_autovacuum_min_duration           | 600000ms   | sighup    | -1    | 2147483647
autovacuum_analyze_threshold          | 50         | sighup    | 0     | 2147483647
autovacuum_vacuum_threshold           | 50         | sighup    | 0     | 2147483647
autovacuum_max_workers                | 3          |postmaster| 1     | 262143
autovacuum_vacuum_cost_delay          | 2ms        | sighup    | -1    | 100
autovacuum_multixact_freeze_max_age   |20000000000|postmaster| 10000 | 9223372036854775807
autovacuum_work_mem                   | -1kB       | sighup    | -1    | 2147483647
autovacuum_freeze_max_age             |10000000000|postmaster| 100000| 9223372036854775807
autovacuum_vacuum_insert_threshold    | 1000       | sighup    | -1    | 2147483647
autovacuum_vacuum_cost_limit          | -1         | sighup    | -1    | 10000
autovacuum_vacuum_insert_scale_factor| 0.2        | sighup    | 0     | 100
autovacuum_vacuum_scale_factor        | 0.2        | sighup    | 0     | 100
autovacuum_analyze_scale_factor       | 0.1        | sighup    | 0     | 100
(15 rows)
```

Настройка автовакуума

- для минимизации простоя автовакуума `autovacuum_naptime` **стоит установить в минимальное значение: 1 секунду**
- `log_autovacuum_min_duration` стоит подобрать значение (по аналогии с `checkpoint_warnings`), при котором сообщения в диагностический лог кластера будут выводиться не часто
- `autovacuum_vacuum_cost_limit` по умочанию -1, что означает что он равен vacuum_cost_limit, который по умолчанию равен 200
  - делится на всех autovacuum workers
  - по достижению лимита каждый рабочий процесс автовакуума засыпает на время в диапазоне: от `autovacuum_cost_delay` до его четырёхкратного значения
  - параметр можно установить на уровне таблицы
- **`autovacuum_vacuum_cost_delay` стоит установить в ноль на уровне кластера и не устанавливать на уровне таблиц**
  - для команды VACUUM `vacuum_cost_delay` по умолчанию ноль

Параметр `autovacuum_naptime`

- для минимизации простоя автовакуума `autovacuum_naptime` стоит установить в минимальное значение `'1s'`
- если автовакууму не удалось заблокировать таблицу, он ее пропускает. Следующая попытка будет в новом цикле автовакуума
- при **`autovacuum_naptime = '1s'`** попытки через в секунду после окончания предыдущего цикла вакуумирования:

```
22:29:07.466 [195773] LOG:  statement: CREATE INDEX ON test(id);
22:29:07.652 [195796] LOG:  skipping vacuum of "test" --- lock not available
22:29:08.653 [195797] LOG:  skipping vacuum of "test" --- lock not available
22:29:09.652 [195798] LOG:  skipping vacuum of "test" --- lock not available
22:29:10.653 [195799] LOG:  skipping vacuum of "test" --- lock not available
```

- При **`autovacuum_naptime = '3s'`** попытки раз в три секунды:

```
22:33:47.021 [195773] LOG:  statement: CREATE INDEX ON test(id);
22:33:49.366 [195978] LOG:  skipping vacuum of "test" --- lock not available
22:33:52.373 [195981] LOG:  skipping vacuum of "test" --- lock not available
22:33:55.373 [195985] LOG:  skipping vacuum of "test" --- lock not available
22:33:58.378 [195991] LOG:  skipping vacuum of "test" --- lock not available
```

Выбор таблиц автовакуумом

- таблица вакуумируется, если:
  - доля мертвых строк больше scale-фактора для вакуума: `pg_stat_all_tables.n_dead_tup/pg_class.reltuples > autovacuum_vacuum_scale_factor`
  - доля вствленных с последнего вакуумирования трок больше scale-фактора для вставок
- статистика по таблице обновится автовакуумом, если:
  - доля изменившихся строк с последнего анализа больше, чем scale-фактор для анализа:
  - `pg_stat_all_tables.n_mod_since_analyze/pg_class.reltuples > autovacuum_analyze_scale_factor`
- значения параметров `autovacuum_vacuum_threshold`, `autovacuum_vacuum_insert_threshold`, `autovacuum_analyze_scale_factor` можно не учитывать, так как таблицы с небольшим числом строк редко требуют внимания

Рекомендации по настройке автовакуума

- возможность очистить старые версии строк в блоках определяется горизонтом базы данных
- если горизонт не сдвигается долгое время, то ни HOT cleanup, ни автовакуум не смогут очистить место в блоках от старых версий строк
- лучше уменьшать задержки в циклах процессов автовакуума, чем увеличивать число рабочих процессов автовакуума

Важность наблюдения за горизонтом баз данных

- наблюдать за горизонтом баз данных нужно для поиска причин, по которым он удерживается
- горизонт удерживают запросы на репликах, если включена обратная связь (`hot_standby_feedback=on`)
- автовакуум стоит настраивать после выбора `checkpoint_timeout` и `shared_buffres`
- время обработки базы данных автовакуумом желательно чтобы было не длиннее времени до горизонта
- автовакуум обрабатывает одну таблицу в одной транзакции, она не должна удерживать горизонт (быть самой длинной по времени)
  - на таких таблицах стоит уменьшить `autovacuum_vacuum_scale_factor`
- уменьшать время обработки баз данных можно уменьшая этот параметр на уровне кластера
- если уменьшить не удается, то увеличивать `autovacuum_max_workers`

Мониторинг горизонта баз данных

- горизонт баз данных кластера в количестве номеров транзакций, отстоящих от текущей:

```
select datname, greatest(max(age(backend_xmin)), max(age(backend_xid))) from pg_stat_activity
where backend_xmin is not null or backend_xid is not null group by datname order by datname;
```

- длительность самого долгого запроса или транзакции, который и удерживает горизонт:

```
select datname, extract(epoch from max(clock_timestamp()-xact_start)) from pg_stat_activity
where backend_xmin is not null or backend_xid is not null group by datname order by datname;
```

- удержание горизонта (удерживают на всех базах) физическими слотами репликации, если включена обратная связь (hot_standby_feedback=on

```
select max(age(xmin)) from pg_replication_slots;
select backend_xmin, application_name from pg_stat_replication order by  age(backend_xmin) desc;
```

- в самих репликах искать процессы, выполняющие команды, удерживающие горизонт можно так же, как и на мастере - запросом к `pg_stat_activity`

```
select age(backend_xmin), extract(epoch from (clock_timestamp()-xact_start)) secs, pid, datname database,
state from pg_stat_activity where backend_xmin IS NOT NULL OR backend_xid IS NOT NULL order by
greatest(age(backend_xmin), age(backend_xid)) desc;
```

Параметры автовакуума на уровне таблиц

- определить, какие параметры установлены, можно запросом:

```
select nspname, relname, reloptions from pg_class join pg_namespace ns on relnamespace
= ns.oid where reloptions is not null and relkind in ('r','t','m','p') order by 2;
     nspname     |     relname     |                      reloptions
-----------------+-----------------+-----------------------------------------------
 pg_toast_temp_0 | pg_toast_16394  | {vacuum_truncate=false}
 pg_toast        | pg_toast_16399  | {autovacuum_vacuum_cost_delay=23}
 pg_temp_0       | reloptions_test | {vacuum_truncate=false,autovacuum_enabled=false}
 public          | test            | {autovacuum_enabled=off}
```

- вернуть значения по умолчанию:

```
alter table test reset (autovacuum_enabled , fillfactor, ..| all );
```

- установить значения:

```
alter table test set (fillfactor=100,..)
```

Параметр `default_statistics_target`

- устанавливает
  - число наиболее часто встречающихся значений в столбцах таблиц (`pg_stats.most_common_vals`)
  - количество корзин в гистрограммах распределения значений в столбцах (`pg_stats.histogram_bounds`)
  - число строк (`default_statistics_target * 300`) случайной выборки по которым собирается статистика
- по умолчанию 100
- максимальное значение 10000
- можно установить для конкретного столбца таблицы или индекса по выражению:

```
alter table test alter column id set statistics 10000;
alter index test alter column  1 set statistics 10000;
```

Раздувание (bloat) таблиц и индексов

- автовакуум может не обработать таблицу из-за того, что:
  - горизонт базы данных долго не сдвигался
  - в момент обращения автовакуума к таблице на ней была установлена несовместимая с автовакуумом блокировка
- после того как автовакуум отработает, размеры файлов вряд ли уменьшатся
- блоки будут использоваться в будущем под новые версии строк
- слишком часто заниматься мониторингом не нужно, более актуальна проверка свободного места на дисках
- пример, как может выглядеть отчёт:

```
Is N/A |      Table       | Size | Extra         |Bloat estimate|  Live  |    Last Vaccuum    |Fillfactor
-------+------------------+------+---------------+--------------+--------+--------------------+----------
       | pgbench_history |2832kB|~16 kB (0.56%)|~16 kB (0.56%)|~2816 kB|   11:11:11 (auto)|    100
```

```
create extension pgstattuple;
\dx+ pgstattuple
select relname, b.* from pg_class, pgstattuple_approx(oid) b WHERE relkind='r';
select relname, b.* from pg_class, pgstatindex(oid) b WHERE relkind='i' order by 10;
```

Практика

- Часть 1. Параметры команды `vacuum`
- Часть 2. Наблюдение за вакуумом
- Часть 3. Расширение для просмотра карты видимости и заморозки `pg_visibility`
- Часть 4. Интервал между циклами автовакуума, параметр `autovacuum_naptime`
- Часть 5. Сравнение вакуума 17 версии с предыдущими версиями
- Часть 6. Число сканирований индексов при вакуумировании
- Часть 7. Логирование автовакуума
- Часть 8. Расширение `pgststtuple`
- Часть 9: Условие обработки индексов: 2% строк

13

Использование диагностического журнала

Диагностический журнал

- `logging_collector` (по умолчанию `off`) рекомендуется установить значение в `on`
- запустится фоновый процесс logger, который собирает (collect) сообщения, отправленные в stderr и записывает их в файлы лога
- `log_min_messages=WARNING` по умолчанию, что означает логирование сообщений с уровнями `ERROR, LOG, FATAL, PANIC`
- `log_min_error_statement=`ERROR по умолчанию. Минимальный уровень важности для команд SQL, которые завершились с ошибкой
- `log_directory=log` (PGDATA/log) по умолчанию. Задаёт путь к директории файлов лога, можно поменять на точку монтирования, скорость и объем которых достаточны для приема логов. Есть параметры, настраивающие ротацию файлов лога

```
psql -c "alter system set logging_collector = on;"
sudo systemctl restart tantor-se-server-16
ps -ef | grep logger
postgres   21861   21860  0 09:37  00:00:00 postgres: logger
```

Параметры диагностики

- параметров конфигурации логирования больше 35
- также есть больше 8 параметров для отладки команд SQL
- расширения могут иметь параметры для логирования
- основные параметры:

```
alter system set logging_collector=on;
alter system set log_min_duration_statement='8s';
alter system set log_statement=ddl;
alter system set log_min_error_statement=ERROR;
alter system set log_temp_files='1MB';
alter system set cluster_name='main';
alter system set log_autovacuum_min_duration='10s';
alter system set log_disconnections=on;
alter system set log_connections=on;
alter system set log_lock_waits=true;
alter system set deadlock_timeout='60s';
alter system set log_recovery_conflict_waits=on;
select pg_reload_conf();
```

```
\dconfig *debug*
debug_assertions                          | off
debug_discard_caches                      | 0
debug_io_direct                           |
debug_logical_replication_streaming       | buffered
debug_parallel_query                      | off
debug_pretty_print                        | on
debug_print_parse                         | off
debug_print_plan                          | off
debug_print_rewritten                     | off
jit_debugging_support                     | off
(10 rows)
\dconfig log*
log_autovacuum_min_duration               | 10min
log_checkpoints                           | on
log_connections                           | off
log_disconnections                        | off
log_duration                              | off
log_error_verbosity                       | default
log_executor_stats                        | off
logging_collector                         | off
log_lock_waits                            | off
log_file_mode                             | 0600
log_filename                              | postgresql-
%Y-%m-%d_%H%M%S.log
logging_collector                         | on
log_hostname                              | off
logical_decoding_work_mem                 | 64MB
log_line_prefix                           | %m [%p]
log_lock_waits                            | off
log_min_duration_sample                   | -1
log_min_duration_statement                | -1
```

Отслеживание использования временных файлов

- `cluster_name = 'main'` По умолчанию пусто. Рекомендуется установить. Значение добавляется к названию процессов экземпляра, что упрощает их идентификацию. На реплике по умолчанию используется для идентификации `wal_receiver`
- `log_temp_files='1MB'` (по умолчанию отключено) логирует имена и размеры создаваемых временных файлов **в момент их удаления**
- при нулевом значении логируются файлы любого размера
- временные файлы создаются в директории табличных пространств, указанных в параметре `temp_tablespaces`
  - можно ограничить параметром `temp_file_limit`
  - рекомендуется установить `log_temp_files` и `temp_file_limit`
- пример сообщения о том, что временный файл дорос до 97Мб:

```
STATEMENT:  CREATE INDEX ON test(id);
LOG:  temporary file: path "base/pgsql_tmp/pgsql_tmp137894.0.fileset/0.0",
size 101810176
```

Отслеживание работы автовакуума и автоанализа

- `log_autovacuum_min_duration`, по умолчанию установлен в 10 минут. Если автовакуум превысит это время при обработке таблицы, то в лог кластера запишется сообщение. При возникновении таких сообщений стоит выяснять причину долгого вакуумирования таблицы

```
LOG:  automatic vacuum of table "postgres.public.test": index scans: 37
pages: 0 removed, 88496 remain, 88496 scanned (100.00% of total)
tuples: 10000000 removed, 10000000 remain, 0 are dead but not yet removable
removable cutoff: 799, which was 0 XIDs old when operation ended
new relfrozenxid: 798, which is 2 XIDs ahead of previous value
frozen: 1 pages from table (0.00% of total) had 82 tuples frozen
index scan needed: 44249 pages from table (50.00% of total) had 10000000 dead item identifiers removed
index "test_id_idx": pages: 54840 in total, 0 newly deleted, 0 currently deleted, 0 reusable
avg read rate: 518.021 MB/s, avg write rate: 47.473 MB/s
buffer usage: 385747 hits, 1864788 misses, 170895 dirtied
WAL usage: 231678 records, 83224 full page images, 248448578 bytes
system usage: CPU: user: 25.72 s, system: 0.38 s, elapsed: 28.12 s

LOG:  automatic analyze of table "postgres.public.test"
avg read rate: 498.808 MB/s, avg write rate: 0.018 MB/s
buffer usage: 2906 hits, 27199 misses, 1 dirtied
system usage: CPU: user: 0.42 s, system: 0.00 s, elapsed: 0.42 s
```

Наблюдение за контрольными точками

- первая запись передаётся в лог, когда начинается контрольная точка
- total = 09:31:35.070 - 09:27:05.095
  - примерно соответствует 270 секундам, которые получаются перемножением checkpoint_completion_target * checkpoint_timeout (0.9*300=270)
- total=write+sync время записи в WAL-файлы
- sync= время затраченное на вызовы fdatasync по WAL-файлам

```
09:27:05.095 LOG:  checkpoint starting: time
09:31:35.070 LOG: checkpoint complete: wrote 4315 buffers (26.3%); 0 WAL file(s) added, 0 removed,
6 recycled; write=269.938 s, sync=0.009 s, total=269.976 s; sync files=15, longest=0.003 s, average=0.001 s;
distance=109699 kB, estimate=109699 kB; lsn=8/1164B2E8, redo lsn=8/BC98978
```

- sync files=15 (синхронизировано файлов) - число файлов данных (в табличных пространствах, чьи блоки располагаются в буферном кэше) в которые велась запись и по ним в конце контрольной точке посылался вызов fsync
- longest=0.003 s (самая_долгая синхр.) - наибольшая длительность обработки одного файла

```
LOG:  checkpoint complete: wrote 8596 buffers (52.5%); 0 WAL file(s) added, 0 removed, 33 recycled;
write=25.057 s, sync=9.212 s, total=35.266 s; sync files=4, longest=9.181 s, average=2.303 s;
distance=540552 kB, estimate=550280 kB;  lsn=9/16BC03F0, redo lsn=8/F82504B0
```

Описание записей log_checkpoints

- `wrote 4315 buffers` число грязных буферов, которые записаны по контрольной точке. Одновременно с checkpointer грязные блоки могут записывать серверные процессы и bgwriter
- `(26.3%)` процент от общего количества буферов буферного кэша, задаваемых параметром `shared_buffers`
- `file(s) added, 0 removed, 6 recycled` число созданных, удалённых, повторно использованных WAL сегментов
- `distance=109699 kB` (расстояние) - объем записей WAL между **началом предыдущей** контрольной точки и **началом текущей**

```
09:22:05.087 LOG: checkpoint starting: time
09:26:35.066 LOG: checkpoint complete: wrote 3019 buffers (18.4%); 0 WAL file(s) added, 0 removed, 6 recycled;
write=269.951 s, sync=0.009 s, total=269.980 s; sync files=14, longest=0.004 s, average=0.001 s;
distance=99467 kB, estimate=108859 kB; lsn=8/AA004C8, redo lsn=8/5177990
09:27:05.095 LOG: checkpoint starting: time
09:31:35.070 LOG: checkpoint complete: wrote 4315 buffers (26.3%); 0 WAL file(s) added, 0 removed, 6 recycled;
write=269.938 s, sync=0.009 s, total=269.976 s; sync files=15, longest=0.003 s, average=0.001 s;
distance=109699 kB, estimate=109699 kB; lsn=8/1164B2E8, redo lsn=8/BC98978
```

Описание записей `log_checkpoints`

- `estimate=109699 kB` (расстояние которое ожидалось) рассчитывается, чтобы оценить сколько WAL сегментов будет использовано в следующей контрольной точке
- если нули в "`0 WAL file(s) added, 0 removed`", то оценка `estimate` верная. Сколько файлов удалить определяется параметрами `min_wal_size, max_wal_size, wal_keep_size, max_slot_wal_keep_size, wal_init_zero=on, wal_recycle=on`

```
09:22:05.087 LOG: checkpoint starting: time
09:26:35.066 LOG: checkpoint complete: wrote 3019 buffers (18.4%); 0 WAL file(s) added, 0 removed, 6 recycled;
write=269.951 s, sync=0.009 s, total=269.980 s; sync files=14, longest=0.004 s, average=0.001 s;
distance=99467 kB, estimate=108859 kB; lsn=8/AA004C8, redo lsn=8/5177990
09:27:05.095 LOG: checkpoint starting: time
09:31:35.070 LOG: checkpoint complete: wrote 4315 buffers (26.3%); 0 WAL file(s) added, 0 removed, 6 recycled;
write=269.938 s, sync=0.009 s, total=269.976 s; sync files=15, longest=0.003 s, average=0.001 s;
distance=109699 kB, estimate=109699 kB; lsn=8/1164B2E8, redo lsn=8/BC98978
```

Утилита pg_waldump и записи log_checkpoints

- для просмотра записей в WAL-файлах используется утилита pg_waldump. По умолчанию утилита ищет WAL-файлы в "." (текущей директории откуда она запущена), дальше в ./pg_wal, $PGDATA/pg_wal
- в логе и выводе pg_controldata в LSN ведущие нули после "/" не печатаются
- в выводе pg_waldump в lsn и prev ноль печатется, а в redo не печатается

```
pg_controldata | grep check | head -n 3
Latest checkpoint location:           8/1164B2E8
Latest checkpoint's REDO location:    8/0BC98978
Latest checkpoint's REDO WAL file:    000000010000000080000000B
```

```
pg_waldump -s 8/0B000000 | grep CHECKPOINT
rmgr: XLOG len (rec/tot): 148/148, tx: 0,
lsn: 8/1164B2E8, prev 8/1164B298, desc: CHECKPOINT_ONLINE redo 8/0BC98978;
tli 1; prev tli 1; fpw true; xid 8064948; oid 33402; multi 1; offset 0; oldest xid 723 in DB 1;
oldest multi 1 in DB 5; oldest/newest commit timestamp xid: 0/0; oldest running xid 8064947; online
pg_waldump: error: error in WAL record at 8/1361C488: invalid record length at 8/1361C4B0:
expected at least 26, got 0
```

```
09:27:05.095 LOG:  checkpoint starting: time
09:31:35.070 LOG: checkpoint complete: wrote 4315 buffers (26.3%); 0 WAL file(s) added, 0 removed,
6 recycled; write=269.938 s, sync=0.009 s, total=269.976 s; sync files=15, longest=0.003 s, average=0.001 s;
distance=109699 kB, estimate=109699 kB; lsn=8/1164B2E8, redo lsn=8/0BC98978
```

Утилита pg_waldump и записи log_checkpoints

- lsn 8/1164B2E8 запись о конце контрольной точки
- redo 8/0BC98978 запись о начале контрольной точки, с которой начнется восстановление в случае сбоя экземпляра
- prev 8/1164B298 адрес начала предыдущей записи в журнале
- distance объем журнала от начала предыдущей до начала завершенной контрольной точки '8/0BC98978'::pg_lsn-'8/05177990'::pg_lsn

```
pg_controldata | grep check | head -n 3
Latest checkpoint location:           8/1164B2E8
Latest checkpoint's REDO location:    8/BC98978
Latest checkpoint's REDO WAL file:    000000010000000080000000B
```

```
pg_waldump -s 8/0B000000 | grep CHECKPOINT
rmgr: XLOG len (rec/tot): 148/148, tx: 0,
lsn: 8/1164B2E8, prev 8/1164B298, desc: CHECKPOINT_ONLINE redo 8/BC98978;...
```

```
09:26:35.066 LOG: checkpoint complete: wrote 3019 buffers (18.4%); 0 WAL file(s) added, 0 removed, 6 recycled;
write=269.951 s, sync=0.009 s, total=269.980 s; sync files=14, longest=0.004 s, average=0.001 s;
distance=99467 kB, estimate=108859 kB; lsn=8/AA004C8, redo lsn=8/5177990
09:27:05.095 LOG:  checkpoint starting: time
09:31:35.070 LOG: checkpoint complete: wrote 4315 buffers (26.3%); 0 WAL file(s) added, 0 removed,
6 recycled; write=269.938 s, sync=0.009 s, total=269.976 s; sync files=15, longest=0.003 s, average=0.001 s;
distance=109699 kB, estimate=109699 kB; lsn=8/1164B2E8, redo lsn=8/BC98978
```

Диагностика частоты соединений с базой данных

- `log_disconnections=on` записывает в лог событие завершения сессии. Записывается та же информация, что `log_connections` плюс длительность сессии.
  - преимущество в том, что выводится одна строка, что не замусоривает лог
  - позволяет идентифицировать короткие по времени сессии
  - пример сообщения о длительности сессии сессии 4 секунды:

```
LOG: disconnection: session time: 0:00:04.056 user=oleg database=db1 host=[vm1]
```

- `log_connections=on` записывает в лог попытки установить сессию
  - недостаток в том, что неудачные попытки отличаются только дополнительной строкой:

```
LOG: connection received: host=[local]
LOG: connection authorized: user=postgres database=db2 application_name=psql
FATAL: database "db2" does not exist
LOG: connection received: host=[local]
LOG: connection authorized: user=alice database=alice application_name=psql
FATAL: role "alice" does not exist
```

Диагностика блокирующих ситуаций

- `log_lock_waits=true`. По умолчанию отключен. Рекомендуется включить, чтобы получать сообщения в диагностический журнал что какой-либо процесс ждет дольше, чем: `deadlock_timeout`
- `deadlock_timeout='60s'`. По умолчанию 1 секунда, что слишком мало и на нагруженных экземплярах создает значительные издержки
- `log_startup_progress_interval='10s'` не стоит отключать
- `log_recovery_conflict_waits=on`. По умолчанию `off`. Процесс `startup` запишет сообщение в лог реплики, если не сможет применить WAL к реплике дольше, чем `deadlock_timeout`

```
LOG: recovery still waiting after 60.555 ms: recovery conflict on lock
DETAIL: Conflicting process: 5555.
```

- Наличие конфликтов можно увидеть в представлении (мало деталей):

```
select * from pg_stat_database_conflicts where datname='postgres';
datid|datname |tblspc|confl_lock|confl_snapshot|confl_bufferpin|deadlock
-----+--------+------+----------+--------------+---------------+--------
13842|postgres|     0 |         0 |             1 |              1 |       0
```

Практика

- Часть 1. Чтение сообщений вакуума и автовакуума
- Часть 2. Чтение сообщений о контрольной точке
- Часть 3. Чтение сообщений о контрольной точке pg_waldump
- Часть 4. Размер директории PGDATA/pg_wal

14

Накопительная статистика

Накопительная статистика

- накапливает статистику о работе экземпляра. Сбор статистики создает накладные расходы, поэтому есть параметры, которыми можно включить сбор дополнительной статистики
- параметры которые включены не стоит отключать, так как их результат используется процессами экземпляра

```
 select name, setting, unit, context, min_val, max_val from pg_settings where
name like 'track%';
            name              | setting | unit |  context   | min_val | max_val
------------------------------+---------+------+------------+---------+---------
 track_activities             | on      |      | superuser  |         |
 track_activity_query_size    | 1024    | B    | postmaster | 100     | 1048576
 track_commit_timestamp       | off     |      | postmaster |         |
 track_counts                 | on      |      | superuser  |         |
 track_functions              | none    |      | superuser  |         |
 track_io_timing              | off     |      | superuser  |         |
 track_wal_io_timing          | off     |      | superuser  |         |
(7 rows)
```

Утилита pg_test_timing

- позволяет оценить издержки на получение времени
- любая статистика с временем - следствие обращения к счетчику времени
- пример накладных расходов команды EXPLAIN ANALYZE, которая делает 2 замера времени на каждую строку:

```
CREATE TABLE t AS SELECT * FROM generate_series(1,100000);
\timingcv
SELECT COUNT(*) FROM t;
Time: 15.526 ms
EXPLAIN ANALYZE SELECT COUNT(*) FROM t;
Time: 310.682 ms
```

```
pg_test_timing
Testing timing overhead for 3 seconds.
Per loop time including overhead: 1424.21 ns
Histogram of timing durations:
  < us   % of total      count
     1      0.03043        641
     2     64.30326    1354500
     4     35.46601     747065
     8      0.01234        260
    16      0.09889       2083
```

- задержка зависит от используемого счетчика времени
- задержку измеряет утилита командной строки pg_test_timing
- наиболее быстрый счетчик tsc:

```
cat /sys/devices/system/clocksource/clocksource0/current_clocksource
tsc
```

Просмотр статистики работы процессов

- статистика передается серверными процессами в общую память перед простоем, то есть после выполнения команды и не чаще, чем раз в секунду
- суперпользователи и роли pg_read_all_stats, pg_monitor видят данные обо всех сессиях. Остальные пользователи видят по своей сессии, по остальным сессиям в полях null
- для просмотра накопительной статистики используется более 30 представлений и более 100 функций с названием pg_stat*
- представление pg_stat_database содержит по одной строке для каждой базы данных:

```
 select datname, numbackends, sessions, round(100.0*blks_hit/NULLIF(blks_read+blks_hit,0),2) hitratio,
temp_files, temp_bytes, round((100.0*active_time/NULLIF(session_time+active_time, 0))::numeric,2) activeratio,
idle_in_transaction_time idleintrans, blk_read_time, blk_write_time from pg_stat_database;
   datname   | numbackends | sessions | hitratio | temp_files | temp_bytes | activeratio | idleintrans | blk_read_time | blk_write_time
-------------+-------------+----------+----------+------------+------------+-------------+-------------+---------------+----------------
             |           0 |        0 |    99.75 |          0 |          0 |             |           0 |             0 |              0
  postgres   |           1 |       13 |    97.96 |          1 |    1400000 |        0.07 |   36177.453 |      8194.412 |          9.947
  my_test_db |           0 |        0 |          |          0 |          0 |             |           0 |             0 |              0
  template1  |           0 |        0 |          |          0 |          0 |             |           0 |             0 |              0
  template0  |           0 |        0 |          |          0 |          0 |             |           0 |             0 |              0
```

Представление pg_stat_database

- представление pg_stat_database содержит по одной строке для каждой базы данных
- если растет число xact_rollback, то это может указывать на ошибки в коде приложения
- если имеются deadlocks, то это указывает на ошибки в логике приложения

```
 select datname, round(100*xact_commit::numeric/nullif(xact_commit+xact_rollback,0),2) as cratio,
xact_rollback rollbacks, deadlocks, conflicts, temp_files, pg_size_pretty(temp_bytes) as tempsize
from pg_stat_database;
  datname | cratio | rollbacks | deadlocks | conflicts | temp_files | tempsize
 ---------+--------+-----------+-----------+-----------+------------+----------
          |        |         0 |         0 |         0 |          0 | 0 bytes
 postgres | 100.00 |        10 |         0 |         0 |          0 | 0 bytes
```

- Процент попадания должен быть не меньше 90%

```
 select datname, numbackends, sessions, round(100.0*blks_hit/NULLIF(blks_read+blks_hit,0),2) hitratio,
temp_files, temp_bytes, round((100.0*active_time/NULLIF(session_time+active_time, 0))::numeric,2)
activeratio, idle_in_transaction_time idleintrans, blk_read_time, blk_write_time from pg_stat_database;
   datname   | numbackends | sessions | hitratio | temp_files | temp_bytes | activeratio | idleintrans | blk_read_time | blk_write_time
 ------------+-------------+----------+----------+------------+------------+-------------+-------------+---------------+----------------
             |           0 |        0 |    99.75 |          0 |          0 |             |           0 |             0 |              0
  postgres   |           1 |       13 |    97.96 |          1 |    1400000 |        0.07 |   36177.453 |      8194.412 |          9.947
```

Прогресс выполнения команд

- Шесть представлений содержат данные о выполняющихся командах:
- pg_stat_progress_analyze - `analyze` и автоанализа
- pg_stat_progress_basebackup - утилиты `pg_basebackup`
- pg_stat_progress_cluster - `cluster` и `vacuum full`
- pg_stat_progress_copy - `copy`
- pg_stat_progress_create_index - `create index` и `reindex`
- pg_stat_progress_vacuum
- можно оценить через какое время команда выполнится и наблюдать за фазами выполнения команд

```
pg_basebackup -D $HOME/backup/1 -P --max-rate=1000
28130/4564784 kB (0%), 0/1 tablespace
select * from pg_stat_progress_basebackup \watch 5
pid|          phase          |backup_total|backup_streamed|tablespaces_total|tablespaces_streamed
---+-------------------------+------------+---------------+-----------------+--------------------
414|streaming database files| 4674334720 |      6502912  |               1 |                  0
```

Представление pg_stat_io

- статистика ввода-вывода
- статистика по каждому типу процесса
- в столбце object могут быть значения:
  - relation (объект постоянного хранения)
  - temp relation (временный объект)
- в столбце context могут быть значения:
  - normal
  - bulkread, bulkwrite, vacuum (буферные кольца)
- по этим столбцам удобно группировать:

```
select * from pg_stat_bgwriter\gx
-[ RECORD 1 ]---------+--------
checkpoints_timed     | 151
checkpoints_req       | 2
checkpoint_write_time | 5074064
checkpoint_sync_time  | 250
buffers_checkpoint    | 93237
buffers_clean         | 622
maxwritten_clean      | 0
buffers_backend       | 18716
buffers_backend_fsync | 0
buffers_alloc         | 21328
```

```
select backend_type name, sum(writes) buffers_written, sum(write_time) write_time, sum(writebacks) writebacks,
sum(writeback_time) writeback_time, sum(evictions) evictions, sum(fsyncs) fsyncs, sum(fsync_time) fsync_time from
pg_stat_io group by backend_type having sum(writes)> 0 or sum(writebacks)> 0 or sum(fsyncs)>0 or sum(evictions)>0;
       name        | buffers_written | write_time | writebacks | writeback_time | evictions | fsyncs | fsync_time
-------------------+-----------------+------------+------------+----------------+-----------+--------+-----------
 client backend    |               0 |          0 |          0 |              0 |       370 |      0 |          0
 autovacuum worker |               0 |          0 |          0 |              0 |        57 |      0 |          0
 background writer |             622 |      6.401 |        576 |         83.795 |           |      0 |          0
 checkpointer      |           93035 |  1386.3310 |      93028 |        323.601 |           |    304 |    215.698
```

Статистики `buffers_backend_fsync` и `fsyncs`

- статистики `pg_stat_bgwriter.buffers_backend_fsync` и `pg_stat_io.fsyncs, pg_stat_io.fsync_time` по `backend_type='client backend'` показывает сколько вызовов fsync выполняли серверные процессы
- если статистики больше нуля
  - стоит проверить статистики и параметры процесса bgwriter
  - производительность ввода-вывода может быть низкой и являться узким местом
  - если `pg_stat_bgwriter.maxwritten_clean` имеет большое значение, то это указывает на то, что стоит увеличивать `bgwriter_lru_maxpages`

Строки представления pg_stat_io

```
select backend_type, left(object,4) obj, context, writes w, round(write_time::numeric,2) wt, writebacks wb,
round(writeback_time::numeric,2) wbt, extends ex, round(extend_time::numeric,2) et, evictions ev, reuses ru,
fsyncs fs, round(fsync_time::numeric,2) fst from pg_stat_io;
```

backend_type	object	context	w	wt	wb	wbt	ex	et	ev	ru	fs	fst
autovacuum launcher	relation	bulkread	0	0.00	0	0.00			0	0		
autovacuum launcher	relation	normal	0	0.00	0	0.00			0		0	0.00
autovacuum worker	relation	bulkread	0	0.00	0	0.00			0	0		
autovacuum worker	relation	normal	0	0.00	0	0.00	3	0.085	0		0	0.00
autovacuum worker	relation	vacuum	0	0.00	0	0.00	0	0	0	0		
client backend	relation	bulkread	0	0.00	0	0.00			0	0		
client backend	relation	bulkwrite	0	0.00	0	0.00	6	0.136	0	0		
client backend	relation	normal	0	0.00	0	0.00	165	1.967	0		0	0.00
client backend	relation	vacuum	24	0.24	0	0.00	0	0	0	24		
client backend	temp relation	normal	50	0.39			151	1.567	51			
background worker	relation	bulkread	0	0.00	0	0.00			0	0		
background worker	relation	bulkwrite	0	0.00	0	0.00	0	0	0	0		
background worker	relation	normal	0	0.00	0	0.00	0	0	0		0	0.00
background worker	relation	vacuum	0	0.00	0	0.00	0	0	0	0		
background worker	temp relation	normal	0	0.00			0	0	0			
background writer	relation	normal	0	0.00	0	0.00					0	0.00
checkpointer	relation	normal	94	4.76	94	2.70					62	160.88
standalone backend	relation	bulkread	0	0.00	0	0.00			0	0		
standalone backend	relation	bulkwrite	0	0.00	0	0.00	0	0	0	0		
standalone backend	relation	normal	0	0.00	0	0.00	0	0	0		0	0.00
standalone backend	relation	vacuum	0	0.00	0	0.00	0	0	0	0		
startup	relation	bulkread	0	0.00	0	0.00			0	0		
startup	relation	bulkwrite	0	0.00	0	0.00	0	0	0	0		
startup	relation	normal	0	0.00	0	0.00	0	0	0		0	0.00
startup	relation	vacuum	0	0.00	0	0.00	0	0	0	0		
walsender	relation	bulkread	0	0.00	0	0.00			0	0		
walsender	relation	bulkwrite	0	0.00	0	0.00	0	0	0	0		
walsender	relation	normal	0	0.00	0	0.00	0	0	0		0	0.00
walsender	relation	vacuum	0	0.00	0	0.00	0	0	0	0		
walsender	temp relation	normal	0	0.00			0	0	0			

(30 rows)

Характеристики `pg_stat_io`

- основной интерес представляют столбцы writebacks, writeback_time fsyncs, fsync_time так как именно они приводят к передаче системных вызовов в операционную систему
- writebacks выполняются:
  - серверными процессами при расширении файлов (extends) и поиске буфера на вытеснение
  - процессами bgwriter и checkpointer, которые работают только в контексте normal
- fsyncs
  - передаются процессу checkpointer для выполнения в конце контрольной точки один раз по каждому файлу
  - если не могут быть переданы (нет места в структуре памяти, которую очищает checkpointer), то выполняются самим процессом
  - у колец fsyncs не считаются
- у bgwriter и checkpointer в столбцах reads, hits, evictions всегда нули
- у bgwriter и autovacuum launcher в столбце extends нули

Статистики представления `pg_stat_io`

- `evictions` сколько раз процесс очистил буфер для нового блока
  - был ли буфер грязным или чистым не отражается
- `context=bulkwrite,bulkread` означает, что использовался кольцевой буфер
- `extends, extend_time` - время, затраченное на увеличение файлов relations
- `buffers_alloc` сколько раз процессы загружали блоки в буфер, если при этом не использовалось буферное кольцо
- если свободных буферов в кэше буферов нет, то `buffers_alloc` соответствуют `evictions`

```
 select backend_type, context, reads, trunc(read_time) r_time, writes, trunc(write_time) w_time, extends, extend_time,
hits, evictions, reuses from pg_stat_io where reads<>0 or writes<>0 or extends<>0 or hits<>0 or evictions<>0;
    backend_type     | context | reads | r_time | writes | w_time | extends | extend_time |   hits    | evictions | reuses
---------------------+---------+-------+--------+--------+--------+---------+-------------+-----------+-----------+-------
 autovacuum launcher | normal  |     1 |      6 |      0 |      0 |         |             |       590 |         0 |
 autovacuum worker   | normal  |   217 |    310 |      0 |      0 |       2 |       0.038 |    799993 |       101 |
 autovacuum worker   | vacuum  | 47889 |    305 |     13 |      0 |       0 |           0 |   1636676 |       702 | 47171
 client backend      | normal  |  2420 |   2573 |      0 |      0 |   30914 |     764.359 | 459710941 |     12675 |
 client backend      | vacuum  |     1 |      0 |      0 |      0 |       0 |           0 |         3 |         0 |     0
 background writer   | normal  |       |        |   2751 |     21 |         |             |           |           |
 checkpointer        | normal  |       |        | 169856 |   2338 |         |             |           |           |
```

Представления pg_statio_all_tables и pg_statio_all_indexes

- В pg_statio_all_tables статистика чтения блоков таблиц, всех индексов на эту таблицу, TOAST таблицы и её индекса
- В pg_statio_all_indexes статистика по конкретному индексу
  - "hit" означает, что блок находился в буферном кэше
  - "read" означает, что блок отсутствовал в буферном кэше, процессу приходилось очищать буфер (если список свободных блоков пуст), загружать в очищенный буфер блок и только потом читать его содержимое

```
 select schemaname||'.'||relname name, heap_blks_read tabread, heap_blks_hit tabhit, idx_blks_read idxread, idx_blks_hit
idxhit, toast_blks_read toastread, toast_blks_hit toasthit from pg_statio_all_tables order by 2 desc limit 3;
           name           | tabread  |  tabhit   | idxread |  idxhit  | toastread | toasthit
--------------------------+----------+-----------+---------+----------+-----------+----------
 public.pgbench_accounts  | 22702740 | 82951068  |  479143 | 36142117 |           |
 pg_catalog.pg_statistic  |   193579 |  2042126  |    2792 |    41643 |         5 |        1
 public.pgbench_history   |   148559 |  5882517  |         |          |           |
 select schemaname||'.'||relname table, indexrelname index, idx_blks_read idxread, idx_blks_hit idxhit from
pg_statio_all_indexes order by 4 desc limit 3;
           table          |         index          | idxread |  idxhit
--------------------------+------------------------+---------+----------
 public.pgbench_accounts  | pgbench_accounts_pkey  |  483328 | 36178084
 public.pgbench_tellers   | pgbench_tellers_pkey   |    1128 | 13472177
 public.pgbench_branches  | pgbench_branches_pkey  |    1129 |  9042373
```

Представление pg_stat_all_tables

- содержит детальную статистику по таблицам и индексам:

```
 select schemaname||'.'||relname name, seq_scan, idx_scan, idx_tup_fetch, autovacuum_count, autoanalyze_count from
pg_stat_all_tables where idx_scan is not null order by 3 desc limit 1;
          name           | seq_scan | idx_scan | idx_tup_fetch | autovacuum_count | autoanalyze_count
-------------------------+----------+----------+---------------+------------------+-------------------
 public.pgbench_accounts |        0 | 11183162 |      11183162 |             1512 |               266
  select relname name, n_tup_ins ins, n_tup_upd upd, n_tup_del del, n_tup_hot_upd hot_upd, n_tup_newpage_upd
newblock, n_live_tup live, n_dead_tup dead, n_ins_since_vacuum sv, n_mod_since_analyze sa from pg_stat_all_tables
where idx_scan is not null order by 3 desc limit 1;
       name       | ins |   upd   | del | hot_upd | newblock |  live  |  dead   | sv |  sa
------------------+-----+---------+-----+---------+----------+--------+---------+----+------
 pgbench_tellers  |   0 | 5598056 |   0 | 5497197 |   100859 |     10 | 1456051 |  0 |  165
```

- пример теста pgbench если горизонт удерживался и перестал удерживаться:

```
progress: 92100.0 s,   9.4 tps, lat 105.960 ms stddev 4.277, 0 failed
progress: 92200.0 s, 556.1 tps, lat   1.796 ms stddev 2.213, 0 failed
       name       | ins |   upd   | del | hot_upd | newblock |  live  | dead | iv |  ma
------------------+-----+---------+-----+---------+----------+--------+------+----+------
 pgbench_tellers  |   0 | 5762566 |   0 | 5656271 |   106295 |     10 |  124 |  0 | 1201
```

- представление `pg_stat_xact_all_tables` имеет те же столбцы, что и `pg_stat_all_tables`, но показывает только действия, выполненные в текущей транзакции и ещё не попавшие в `pg_stat_all_*`. Столбцы для **`n_live_tup`**, **`n_dead_tup`** и относящиеся к вакуумированию и анализу отсутствуют

Представление pg_stat_all_indexes

- содержат детальную статистику по таблицам и индексам:

```
select schemaname||'.'||relname name, indexrelname, idx_scan, round(extract('epoch' from clock_timestamp() -
 last_idx_scan)) scan_sec, idx_tup_read, idx_tup_fetch from pg_stat_all_indexes order by 3 desc limit 3;
          name            |     indexrelname      | idx_scan | scan_sec | idx_tup_read | idx_tup_fetch
--------------------------+-----------------------+----------+----------+--------------+---------------
 public.pgbench_accounts | pgbench_accounts_pkey |  7080442 |    19489 |     20471311 |       7080442
 public.pgbench_tellers  | pgbench_tellers_pkey  |  2799175 |    19489 |   3740278460 |        124998
 public.pgbench_branches | pgbench_branches_pkey |  2798720 |    19489 |   3049729576 |       2798720
```

- idx_scan число сканирований этого индекса
- last_idx_scan время последнего сканирования этого индекса
- idx_tup_read число индексных записей, возвращённых при сканированиях этого индекса
- idx_tup_fetch актуальных строк таблицы, выбранных с помощью при простом сканировании этого индекса (Index Scan)

Длительность удержания горизонта баз данных

- длительность самого долгого запроса или транзакции всех баз данных кластера:

```
 select datname database, max(now()-xact_start) duration, greatest(max(age(backend_xmin)),
max(age(backend_xid))) age from pg_stat_activity where backend_xmin is not null or backend_xid is not null
group by datname order by datname;
  database |    duration     |  age
----------+-----------------+-------
 postgres | 00:09:41.946193 | 79348
```

- процессы, удерживающие горизонт:

```
 select now()-xact_start duration, age(backend_xmin) age_xmin, age(backend_xid) age_xid, pid, datname,
usename, state, wait_event, left(query,20) query from pg_stat_activity where backend_xmin is not null or
backend_xid is not null order by greatest(age(backend_xmin), age(backend_xid)) desc;
 duration  | age_xmin | age_xid | pid  | datname  | usename  |        state        | wait_event | query
----------+----------+---------+------+----------+----------+---------------------+------------+-------
 00:02:52 |    46732 |         | 8806 | postgres | postgres | idle in transaction | ClientRead | select
 00:00:00 |          |       1 | 8825 | postgres | postgres | active              | WALSync    | END;
 00:00:00 |        1 |         | 9049 | postgres | postgres | active              |            |
```

- можно идентифицировать запросы или транзакции, которые удерживают горизонт и оптимизировать их или перенести запросы на реплики

Представление `pg_stat_wal`

- в представлении одна строка
- `wal_records` сколько журнальных записей записано в WAL-файлы
- `wal_fpi` сколько записано полных образов страниц (full page images)
- `wal_buffers_full` сколько раз вызывалась запись в журналы из-за заполнения журнального буфера
- журнальные буфера вытесняются в страничный кэш linux функцией XLogWrite
- если `wal_sync_method=fdatasync`, `fsync` или `fsync_writethrough`, то функция XLogWrite вызывает функцию issue_xlog_fsync для сброса блоков, относящихся к файлам журнала, из страничного кэша на диск
- Столбцы `wal_write_time` и `wal_sync_time` имеют значения отличные от нуля только, если параметр `track_wal_io_timing=on`

```
select * from pg_stat_wal;
 wal_records | wal_fpi |  wal_bytes  | wal_buffers_full | wal_write | wal_sync |...
-------------+---------+-------------+------------------+-----------+----------+
    33373424 | 1158848 | 10892483009 |                0 |   5736124 |  5730955 |
```

Расширение pg_walinspect

- альтернатива утилите командной строки pg_waldump
- pg_get_wal_stats(lsn1, lsn2, true) выдаёт статистику по журнальным записям
  - Если третий параметр true, то выдаётся статистика ещё и по record_type, а не суммарная по каждому resource_manager
- в примере полные образы блоков (fpi_size) занимают: 83% для таблиц и 76% для индексов. По тесту pgbench tps=14.

```
 select "resource_manager/record_type" type, count, round(count_percentage) "count%", record_size,
round(record_size_percentage) "size%", fpi_size, round(fpi_size_percentage(tantor)) "fpi%", combined_size,
round(combined_size_percentage) "total%" from pg_get_wal_stats((select '0/0'::pg_lsn +
(pg_split_walfile_name(name(oleg))).segment_number * size from pg_ls_waldir() oleg order by name limit 1),
'FFFFFFFF/FFFFFFFF', true) tantor where count<>0 order by 8 desc;
          type          | count | count% | record_size | size% | fpi_size | fpi% | combined_size | total%
------------------------+-------+--------+-------------+-------+----------+------+---------------+--------
 Heap/LOCK              | 16223 |     15 |     1003405 |    11 | 48582684 |   81 |      49586089 |     71
 Btree/INSERT_LEAF      | 16224 |     15 |     1049301 |    11 | 10692280 |   18 |      11741581 |     17
 Heap/UPDATE            | 15822 |     14 |     2865744 |    31 |    12620 |    0 |       2878364 |      4
 Heap/HOT_UPDATE        | 29779 |     27 |     2560740 |    27 |     5432 |    0 |       2566172 |      4
 Heap/INSERT            | 15215 |     14 |     1232365 |    13 |     7620 |    0 |       1239985 |      2
 XLOG/FPI_FOR_HINT      |    89 |      0 |        4539 |     0 |   719100 |    1 |        723639 |      1
 Transaction/COMMIT     | 15331 |     14 |      553376 |     6 |        0 |    0 |        553376 |      1
...
```

Использование расширения `pg_walinspect`

- если горизонт не удерживается, то доля HOT становится больше:

```
          type          |   count | count% | record_size | size% | fpi_size | fpi% | combined_size | total%
------------------------+---------+--------+-------------+-------+----------+------+---------------+--------
 Heap/HOT_UPDATE        | 1125325 |     54 |    96690453 |    62 | 28860128 |   47 |     125550581 |     58
 Heap/INSERT            |  372681 |     18 |    30187111 |    19 |     8764 |    0 |      30195875 |     14
 Heap2/PRUNE            |  131095 |      6 |    11757645 |     8 |  4885592 |    8 |      16643237 |      8
 Heap2/VISIBLE          |   58699 |      3 |     3822780 |     2 | 11975568 |   20 |      15798348 |      7
 XLOG/FPI_FOR_HINT      |    1929 |      0 |       98379 |     0 | 15082820 |   25 |      15181199 |      7
 Transaction/COMMIT     |  375127 |     18 |    13509524 |     9 |        0 |    0 |      13509524 |      6
...
```

- TPS увеличиваются с 14 до 660
- чем ближе горизонт к текущему моменту и чем меньше FILLFACTOR тем больше вероятность того что доля HOT увеличится
- даёт ли уменьшение этих значений эффект можно запросом обратив внимание на долю Heap/HOT_UPDATE
- HOT возможен если:
  - изменения не затрагивают проиндексированных столбцов
  - новая версия строки должна поместиться в тот же блок, что и обновляемая версия строки

Представление pg_stat_activity

- по одной строке для каждого процесса экземпляра
- строки отражают чем занят процесс в момент обращения к строке
- backend_type - название процесса
- state текущее состояние процесса
- wait_type='Activity', это нормальное состояние процесса, он не заблокирован, простаивает и ждет пробуждения в своём основном (Main) цикле
- wait_type='Lock', то процесс ожидает получения блокировки

```
 select left(backend_type, 14) type, left(state,6) state, wait_event_type wait_type, wait_event,
age(backend_xid) age_xid, age(backend_xmin) age_xmin, round(extract('epoch' from clock_timestamp()
xact_start)) sec, substring(query,0, 20) query from pg_stat_activity;
      type      | state  | wait_type |      wait_event      | age_xid | age_xmin | sec  |        query
----------------+--------+-----------+----------------------+---------+----------+------+--------------------
autovacuum lau |        | Activity  | AutoVacuumMain       |         |          |      |
pg_wait_sampli |        | Extension | Extension            |         |          |      |
client backend | idle i | Client    | ClientRead           |   92700 |          | 1010 | lock table t;
client backend | active | Lock      | relation             |         |    92700 |  967 | select * from t lim
client backend | active |           |                      |       1 |    92700 |    0 | UPDATE pgbench_bran
...
```

Блокирующие процессы и функция `pg_blocking_pids()`

- если pg_stat_activity.wait_type='Lock' или pg_locks.granted='f', то процесс заблокирован, то есть ожидает получения блокировки
- `pg_blocking_pids(pid)` показывает процессы-блокировщики
  - функция на короткое время получает легковесную блокировку на все секции структуры блокировок, поэтому не стоит ее вызывать часто

```
 select a.pid blocked, bl.pid blocker, a.wait_event_type wait, a.wait_event event, age(a.backend_xmin) age,
round(extract('epoch' from clock_timestamp() - a.query_start)) waitsec, bl.wait_event_type bl_type,
bl.wait_event bl_wait, round(extract('epoch' from clock_timestamp() - bl.query_start)) bl_sec,
left(a.query,10) blocked_q, left(bl.query,10) blocker_q from pg_stat_activity a join pg_stat_activity bl on
bl.pid = ANY(pg_blocking_pids(a.pid)) where a.wait_event_type='Lock' and
cardinality(pg_blocking_pids(a.pid))>0;
 blocked|blocker|wait| event   |age|waitsec|bl_type| bl_wait   |bl_sec|    blocked_q   | blocker_q
 -------+-------+----+--------+---+-------+-------+----------+------+---------------+----------
  755637| 754580|Lock|relation|  1|    815| Client|ClientRead|  1126|select count(*)| select 1;
```

- для мониторинга можно использовать представление `pg_locks`:

```
 select pl.pid blocked, pl.mode, a.wait_event_type wait, a.wait_event event, age(a.backend_xmin) age_xmin,
round(extract('epoch' from clock_timestamp() - a.query_start)) waitsec, left(a.query,40) blocked_query from
pg_locks pl left join pg_stat_activity a ON pl.pid = a.pid where pl.granted='f' and a.datname =
current_database();
 blocked|      mode       | wait |  event    | age_xmin | waitsec |      blocked_query
 -------+-----------------+------+-----------+----------+---------+-------------------------
   1234 | AccessShareLock | Lock | relation |   447502 |   11077 | select * from t limit 1;
```

pg_cancel_backend() и pg_terminate_backend()

- отменить команду в блокирующей сессии можно функцией `pg_cancel_backend(pid)`
- если сессия простаивает (`wait_event='ClientRead'`), то отменять нечего и функция бесполезна
- сессию и транзакцию в сессии можно завершить функцией `pg_terminate_backend(pid, timeout миллисекунд DEFAULT 0)`

```
begin;
lock table t;
```

```
select 1;
1
```

```
select 1;
FATAL:  terminating connection due to administrator command
server closed the connection unexpectedly
        This probably means the server terminated abnormally
        before or while processing the request.
The connection to the server was lost. Attempting reset: Succeeded.
select 1;
1
```

```
select pg_cancel_backend(124989);
 t
select a.wait_event_type wait,...
(1 row)
select pg_terminate_backend(124989);
pg_terminate_backend
----------------------
 t
select a.wait_event_type wait,...
(0 rows)
```

Практика

- Часть 1. Статистика ввода-вывода в представлении pg_stat_io
- Часть 2. Выполнение fsyncs при остановленном checkpointer
- Часть 3. Тестирование производительности ввода-вывода
- Часть 4. Выбор размера temp_buffers при работе с временными таблицами с помощью pg_stat_io
- Часть 5. Пример анализа статистик работы команды vacuum с кольцом
- Часть 6. Работа bgwriter и сопоставление статистик в представлениях pg_stat_bgwriter и pg_stat_io
- Часть 7. Работа bgwriter на буферном кэше 128Мб и 1Гб
- Часть 8. Использование расширения pg_walinspect
- Часть 9. Наблюдение за блокировками

15

Расширения pg_stat_statements и pg_stat_kcache

Расширение pg_stat_statements

- детальная статистика работы экземпляра с точностью до команд SQL
- для установки нужно загрузить библиотеку и установить расширение:

```
alter system set shared_preload_libraries = pg_stat_statements;
create extension pg_stat_statements;
```

- в расширение входят 3 функции и 2 представления:

```
\dx+ pg_stat_statements
 function pg_stat_statements(boolean)
 function pg_stat_statements_info()
 function pg_stat_statements_reset(oid,oid,bigint,boolean)
 view pg_stat_statements
 view pg_stat_statements_info
```

- команды объединяются в одну строку в `pg_stat_statements`, когда они выполняются одним и тем же пользователем и имеют идентичные структуры запросов, то есть семантически равнозначны, за исключением литералов и переменных подстановки (literal constants)
- Например: `select * from t where id = 'a'` и `select * from t where id = 'b'` объединятся в `select * from t where id = $1`

Конфигурация pg_stat_statements

- в представлении pg_stat_statements_info два поля:
  - dealloc сколько раз в pg_stat_statements отбрасывались записи о редко выполняемых командах. Число отслеживаемых команд задаётся параметром pg_stat_statements.max, по умолчанию 5000.
  - stats_reset момент последнего сброса всей статистики расширения вызовом функции pg_stat_statements_reset() без параметров
- функцией pg_stat_statements_reset(userid, dbid, queryid, minmax_only) можно сбросить не всю, а только часть статистики:
  - по командам выполняемым пользователем, выполняемым в базе данных, по отдельному запросу

```
select pg_stat_statements_reset();
   pg_stat_statements_reset
-------------------------------
 2025-01-01 11:11:11.111111+03
(1 row)
select * from pg_stat_statements_info;
 dealloc |          stats_reset
---------+-------------------------------
       0 | 2025-01-01 11:11:11.111111+03
(1 row)
```

Параметры конфигурации `pg_stat_statements`

- `pg_stat_statements.max` задаёт максимальное число команд, отслеживаемых расширением, то есть, максимальное число строк в представлении `pg_stat_statements`
- `pg_stat_statements.save` определяет, должна ли статистика сохраняться после перезагрузки сервера
- `pg_stat_statements.track` определяет, какие команды будут отслеживаться: top отслеживаются только команды верхнего уровня
- pg_stat_statements.track_planning устанавливает будут ли отслеживаться операции планирования и длительность фазы планирования
- `pg_stat_statements.track_utility` будут ли отслеживаться команды, отличные от `SELECT, INSERT, UPDATE, DELETE, MERGE`

```
select name, setting, context, min_val, max_val from pg_settings where name like 'pg_stat_statements%';
               name                | setting |  context   | min_val |  max_val
-----------------------------------+---------+------------+---------+------------
 pg_stat_statements.max            | 5000    | postmaster | 100     | 1073741823
 pg_stat_statements.save           | on      | sighup     |         |
 pg_stat_statements.track          | top     | superuser  |         |
 pg_stat_statements.track_planning | on      | superuser  |         |
 pg_stat_statements.track_utility  | on      | superuser  |         |
```

Представление `pg_stat_statements`

- представление заполняется функцией pg_stat_statements(true)
- в представлении **49 столбцов**, из них 10 относятся к `jit`
- 5 столбцов с временем выполнения: total_exec_time, min_exec_time, max_exec_time, mean_exec_time, stddev_exec_time
- calls - сколько раз выполнялась команда
- rows - число прочитанных или измененных строк
- 3 столбца относятся к wal: `wal_records, wal_fpi, wal_bytes`
- 6 столбцов к планам выполнения
- 6 столбцов к кэшу буферов
- 10 столбцов к временным файлам и объектам
- 4 столбца с характеристиками команды

```
 select calls, round(total_exec_time) time, round(stddev_exec_time) delta, rows, plans, round(total_plan_time)
ptime, shared_blks_hit hit, shared_blks_read read, shared_blks_dirtied dirty, shared_blks_written write, wal_fpi,
wal_bytes, left(query,20) query from pg_stat_statements order by 2 desc limit 10;
 calls  | time  | delta |  rows  | plans  | ptime |   hit   | read | dirty | write | wal_fpi | wal_bytes |  query
--------+-------+-------+--------+--------+-------+---------+------+-------+-------+---------+-----------+--------
 529554 | 13478 |     0 | 529554 | 529554 | 16851 | 3302629 | 6963 | 34309 |     0 |   38447 | 117746959 | UPDATE
 529554 | 12370 |     0 | 529554 | 529554 | 14759 | 2118690 |    5 |    15 |     0 |       9 |  46780041 | UPDATE
 529554 | 12246 |     0 | 529554 | 529554 | 15316 | 3046701 |    3 |    38 |     0 |      31 |  46927982 | UPDATE
```

Запросы к представлению pg_stat_statements

- `order by total_exec_time desc limit 10` выдаёт "Top SQL" - наиболее нагружающие экземпляр команды
- `where plans<>0` убирает вспомогательные команды типа: `BEGIN, END`
- `where rows/calls>1000 and toplevel=true` запросы, возвращающие клиенту большое число строк
- `temp_blks_written>0` может указывать на недостаток work_mem
- `100*(blk_read_time+blk_write_time)/total_exec_time` доля ввода-вывода в общем времени выполнения запроса
- `total_plan_time>total_exec_time` время на создание плана больше времени выполнения

```
 select round(total_exec_time::numeric, 2) time, calls, round(mean_exec_time::numeric, 2) mean, round((100 *
total_exec_time / sum(total_exec_time::numeric) OVER ())::numeric, 2) "%cpu", left(query,70) query from
pg_stat_statements order by total_exec_time desc limit 10;
     time     | calls  | mean  | %cpu  |                                  query
--------------+--------+-------+-------+-------------------------------------------------------
  2773963.94 | 227303 | 12.20 | 82.86 | UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE bid = $2
   556235.76 | 227303 |  2.45 | 16.62 | UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE tid = $2
     7922.75 | 227303 |  0.03 |  0.24 | UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE aid = $2
     3735.72 | 227302 |  0.02 |  0.11 | INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES ($1,
     3692.31 | 227303 |  0.02 |  0.11 | SELECT abalance FROM pgbench_accounts WHERE aid = $1
```

Примеры запросов к представлению pg_stat_statements

- `blk_read_time+blk_write_time` время, затраченное на ввод-вывод
- `total_exec_time-blk_read_time-blk_write_time` не на ввод-вывод
- `(100*(blk_read_time+blk_write_time)/total_exec_time)` процент времени, затраченного на ввод-вывод
- `(total_exec_time::numeric/calls)` среднее время выполнения команды
- `((total_exec_time-blk_read_time-blk_write_time)::numeric/calls)` среднее время не на ввод-вывод
- `((blk_read_time+blk_write_time)::numeric/calls)` среднее время на ввод-вывод

```
 select (blk_read_time+blk_write_time)::numeric(20,2) io_time, (total_exec_time-blk_read_time-
blk_write_time)::numeric(20,2) non_io_time, (100*blk_read_time+blk_write_time/total_exec_time)::numeric(20,2)
"io_time%", (total_exec_time::numeric/calls)::numeric(20,2) avg_time, ((total_exec_time-blk_read_time-
blk_write_time)::numeric/calls)::numeric(20, 2) avg_non_io_time, ((blk_read_time+blk_write_time)::numeric/
calls)::numeric(20, 2) avg_io_time, left(query,20) query from pg_stat_statements where calls<>0 order by
io_time desc limit 10;
  io_time | non_io_time | io_time_percent | avg_time | avg_non_io_time | avg_io_time |        query
 ---------+-------------+-----------------+----------+-----------------+-------------+----------------------
   273.58 |       58.60 |        27343.66 |   332.19 |           58.60 |      273.58 | create extension pg_
    42.74 |      -42.64 |         4274.07 |     0.10 |          -42.64 |       42.74 | SELECT c.relname, $1
```

Метрики pg_stat_statements

- можно оптимизировать производительность отдельных команд используя команду **explain (analyze, verbose, settings, buffers, wal)**
- для наблюдения за выполнением команд в целом используется pg_stat_statements, события ожидания pg_stat_activity, диагностический журнал кластера
- ограничения pg_stat_activity:
  - не выдает информации о выполняющихся командах
  - не отслеживаются неудачно завершившиеся команды, например, прекратившие выполняться по statement_timeout
- можно вычислить три метрики для каждой кумулятивной статистики:
  - вычислить разницу между значениями кумулятивных статистик и разделить на интервал времени: ΔM/Δt, то есть вычислить производную по времени.
  - делить не на интервал времени, а на разницу в столбце calls (сколько раз выполнялась команда): ΔM/Δc.
  - Долю (процент), которую занимает показатель относительно всех вызовов M/sum(M)=%M

Примеры метрик представления `pg_stat_statements`

- `Δcalls/Δt` - это метрика "QPS" - число вызовов в секунду
- `Δ(total_plan_time+total_exec_time)/Δcalls` - скользящее среднее время выполнения запроса
- `Δrows/Δt` - количество строк, переданных клиентам
- `Δ(shared_blks_hit+shared_blks_read)*8192/Δt` - объем обрабатываемых данных, нагружающих шину процессор-память
- `Δ(shared_blks_hit+shared_blks_read)/Δcalls` - большие значения указывают на то, что запрос имеет потенциал для оптимизации
- `Δwal_bytes/Δt` - команды генерирующие много WAL
- `Δwal_bytes/Δcalls` - объем WAL, сгенерированных командой
- `%time` - доля запроса среди всех:

```
 select calls, round(total_exec_time::numeric, 2) total_time, round(mean_exec_time::numeric, 2) mean_time,
round((100 * total_exec_time / sum(total_exec_time) OVER ())::numeric, 2) "%time", left(query,40) query
from pg_stat_statements order by total_exec_time desc limit 10;
   calls   | total_time   | mean_time | %time |                  query
 ---------+-------------+-----------+-------+-----------------------------------------
  2842766 | 15188975.46 |      5.34 | 75.80 | UPDATE pgbench_branches SET bbalance = b
  2842767 |  3116248.96 |      1.10 | 15.55 | UPDATE pgbench_tellers SET tbalance = tb
```

Расширение pg_stat_kcache

- дополняет pg_stat_statements и зависит от него
- собирает статистику linux, выполняя системный вызов getrusage после выполнении каждой команды
- в отличие от утилит операционной системы, расширение собирает статистики с детальностью до команды
- позволяет различать читался блок с диска или из страничного кэша
- использует два буфера в разделяемой памяти

```
 select * from (select *,lead(off) over(order by off)-off as diff from pg_shmem_allocations) as a where name
like 'pg_%';
           name            |    off    | size  | allocated_size |  diff
---------------------------+-----------+-------+----------------+---------
 pg_stat_statements        | 148162816 |    64 |            128 |     128
 pg_stat_statements hash   | 148162944 |  2896 |           2944 | 2188544
 pg_stat_kcache            | 150351488 |   992 |           1024 |    1024
 pg_stat_kcache hash       | 150352512 |  2896 |           2944 | 1373056
select name, setting, context, min_val, max_val from pg_settings where name like '%kcache%';
             name              | setting |  context   | min_val |  max_val
-------------------------------+---------+------------+---------+------------
 pg_stat_kcache.linux_hz       | 333333  | user       | -1      | 2147483647
 pg_stat_kcache.track          | top     | superuser  |         |
 pg_stat_kcache.track_planning | off     | superuser  |         |
```

Статистики, собираемые `pg_stat_kcache`

Статистики в представлениях `pg_stat_kcache` и `pg_stat_kcache_detail`:

- `reads_blks` reads, in 8K-blocks
- `writes_blks` writes, in 8K-blocks
- `user_time` user CPU time used
- `system_time` system CPU time used
- `minflts` page reclaims (soft page faults)
- `majflts` page faults (hard page faults)
- `nswaps` swaps
- `msgsnds` IPC messages sent
- `msgrcvs` IPC messages received
- `nsignals` signals received
- `nvcsws` voluntary context switches
- `nivcsws` involuntary context switches

```
alter system set shared_preload_libraries = pg_stat_statements, pg_wait_sampling, pg_stat_kcache;
create extension pg_stat_kcache;
\dx+ pg_stat_kcache
 function pg_stat_kcache()
 function pg_stat_kcache_reset()
 view pg_stat_kcache
 view pg_stat_kcache_detail
```

Просмотр статистик `pg_stat_kcache`

- статистика по базам данных в представлении `pg_stat_kcache`:

```
 select datname database, pg_size_pretty(exec_minflts*4096) reclaims, pg_size_pretty(exec_majflts*4096) faults,
pg_size_pretty(exec_reads) reads, pg_size_pretty(exec_writes) writes, round(exec_system_time::numeric,0) sys,
round(exec_user_time::numeric,0) usr,  exec_nvcsws vsw, exec_nivcsws isw from pg_stat_kcache;
  database | reclaims | faults  | reads   | writes | sys | usr | vsw  | isw
----------+----------+---------+---------+--------+-----+-----+------+------
 postgres |   183 MB | 0 bytes | 226 MB | 13 GB  |  39 | 115 | 2717 | 2831
```

- Для соединения с `pg_stat_statements` удобно использовать функцию **`pg_stat_kcache()`**:

```
 select d.datname database, round(s.total_exec_time::numeric, 0) time, s.calls, pg_size_pretty(exec_minflts*4096)
reclaims, pg_size_pretty(exec_majflts*4096) faults, pg_size_pretty(k.exec_reads) reads,
pg_size_pretty(k.exec_writes) writes, round(k.exec_user_time::numeric, 2) user, round(k.exec_system_time::numeric,
2) sys, k.exec_nvcsws vsw, k.exec_nivcsws isw, left(s.query, 18) query from pg_stat_statements s join
pg_stat_kcache() k using (userid, dbid, queryid) join pg_database d on s.dbid = d.oid order by 2 desc;
 database|time   |calls | reclaims|  faults | reads | writes  | user  | sys  | vsw | isw |       query
 --------+-------+------+---------+---------+-------+---------+-------+------+-----+-----+------------------
 postgres| 25407 |    1 | 3752 kB | 0 bytes | 33 MB | 1277 MB | 11.42 | 4.00 | 306 | 362 | insert into test.f
 postgres| 24817 |    1 | 3752 kB | 0 bytes | 15 MB | 1297 MB | 11.36 | 3.92 | 294 | 167 | insert into test.f
 postgres| 24187 |    1 | 3748 kB | 0 bytes | 35 MB | 1321 MB | 11.70 | 3.71 | 277 | 598 | insert into test.f
```

Практика

- Часть 1. Установка расширения pg_stat_kcache
- Часть 2. Использование расширения pg_stat_kcache
- Часть 3. Производительность при использовании direct i/o

16

Расширение pg_wait_sampling

Расширение pg_wait_sampling

- входит во все сборки СУБД Tantor
- выдает статистику по событиям ожиданий всех процессов экземпляра
- для установки нужно загрузить библиотеку и установить расширение:

```
alter system set shared_preload_libraries = pg_stat_statements, pg_stat_kcache, pg_wait_sampling;
create extension if not exists pg_wait_sampling;
```

- библиотека `pg_wait_sampling` должна быть указана позже `pg_stat_statements`, чтобы расширение не перезаписало идентификаторы запросов (queryid)
- расширение использует фоновый процесс `pg_wait_sampling collector`
- процесс опрашивает состояние всех процессов экземпляра
- в расширение входят 4 функции и 3 представления:

```
\dx+ pg_wait_sampling
 function pg_wait_sampling_get_current(integer)
 function pg_wait_sampling_get_history()
 function pg_wait_sampling_get_profile()
 function pg_wait_sampling_reset_profile()
 view pg_wait_sampling_current
 view pg_wait_sampling_history
 view pg_wait_sampling_profile
```

История событий ожидания

- расширение использует "сэмплрование" с частотой от 1 миллисекунды (по умолчанию 10 миллисекунд)
- история доступна через представление:

```
select count(*) from pg_wait_sampling_history;
count
-----
5000
```

- по умолчанию сохраняется история 5000 последних событий ожидания
- расширение также использует разделяемую память под хранение своих трёх структур:
  - очередь (MessageQueue) фиксированного размера 16Кб
  - память под список PID
  - память под идентификаторы команд (`queryid`), выполняющихся процессами

```
 select * from (select *, lead(off) over(order by off)-off as diff from pg_shmem_allocations) as a
where name like '%wait%';
        name        |    off    | size  | allocated_size | diff
--------------------+-----------+-------+----------------+-------
 pg_wait_sampling | 148145920 | 17536 |          17536 | 17536
```

История событий ожидания

- в истории сохраняются события ожиданий всех процессов
- если серверные процессы не сталкиваются с блокировками, то 99.98% событий ожидания будет заполнено ожиданиями фоновых процессов, не связанных с запросами

```
select * from pg_wait_sampling_history where queryid<>0;
 pid   |              ts                | event_type |        event         |       queryid
-------+--------------------------------+------------+----------------------+---------------------
 53718 | 2035-11-11 11:41:55.629229+03 | IO         | BufFileWrite         | 6530354471556151986
 53718 | 2035-11-11 11:41:59.513407+03 | IO         | BufFileRead          | 6530354471556151986
```

- Программы мониторинга могут запрашивать содержимое истории и сохранять его в своём хранилище. Определение максимальной частоты опроса истории:

```
select max(ts), min(ts), max(ts)-min(ts) duration from pg_wait_sampling_history;
             max               |              min               |    duration
-------------------------------+--------------------------------+-----------------
2035-11-11 11:58:19.691753+03 | 2035-11-11 11:55:41.153914+03 | 00:02:38.537839
```

- запрос может использоваться для определения значения параметров `pg_wait_sampling.history_period` и `pg_wait_sampling.history_size`

Параметры расширения `pg_wait_sampling`

- `profile_period` и `history_period` интервал в миллисекундах для выборки из истории событий и для добавления в профиль
  - по умолчанию 10 миллисекунд
  - вероятность поимки события определяется только этими параметрами
  - данные для истории и профиля выбираются и сохраняются независимо друг от друга
- `pg_wait_sampling.history_size` (по умолчанию 5000, максимальное значение определяется типом `int4`) событий в истории
- `profile_pid` и `profile_queries` разбивают в профиле события по процессам и запросам

```
 select name, setting, context, min_val, max_val, boot_val, pending_restart from pg_settings where name
like '%pg_wait_sampling%';
               name                | setting |  context   | min_val |  max_val   | boot_val| pending_restart
-----------------------------------+---------+------------+---------+------------+---------+----------------
 pg_wait_sampling.history_period   | 100000  | superuser | 1       | 2147483647 | 10      | f
 pg_wait_sampling.history_size     | 100     | superuser | 100     | 2147483647 | 5000    | f
 pg_wait_sampling.profile_period   | 10      | superuser | 1       | 2147483647 | 10      | f
 pg_wait_sampling.profile_pid      | on      | superuser |         |            | on      | f
 pg_wait_sampling.profile_queries  | on      | superuser |         |            | on      | f
```

Профиль pg_wait_sampling

- для просмотра числа событий используется представление:

```
select * from pg_wait_sampling_profile where queryid<>0 order by count desc;
  pid  | event_type |     event      |       queryid       | count
-------+------------+----------------+---------------------+-------
 55206 | IO         | DataFileRead   | -650897274631253140 |  1290
 55206 | IO         | DataFileExtend | -877373571429139692 |     6
```

- данные сохраняются по всем процессам, в том числе по серверным процессам завершившихся сессий, которые уже отсутствуют в операционной системе
- если `pg_wait_sampling.profile_pid=true`, то число строк будет неуклонно расти
- **расширение не очищает память, используемую профилем**
- **нужно периодически освобождать память вызовом функции `pg_wait_sampling_reset_profile()`**
  - функция очищает данные, накопленные в профиле событий ожидания
  - функция не очищает данные в истории

Запросы к профилю `pg_wait_sampling`

- профиль показывает события ожидания, если за время от сброса ожидание было поймано хотя бы один раз:

```
select * from pg_wait_sampling_profile where pid=55549;
 pid   | event_type |        event         |       queryid        | count
-------+------------+----------------------+----------------------+--------
 55549 | Client     | ClientRead           |                    0 | 615444
 55549 | IPC        | MessageQueueInternal | 3255387186388375512 |      1
 55549 | IO         | DataFileRead         |                    0 |      6
```

- столбец pid позволяет связать строки с представлением `pg_stat_activity`:

```
select p.pid, left(a.backend_type, 14) process_type, a.application_name app, p.event_type, p.event,
 p.count from pg_wait_sampling_profile p join pg_stat_activity a on p.pid = a.pid
 where event_type<>'Activity';
 pid   |  process_type   |   app    | event_type |         event          | count
-------+-----------------+----------+------------+------------------------+--------
 60577 | client backend  | psql     | Client     | ClientRead             | 398415
 60683 | client backend  | pgbench  | IO         | WALSync                | 246986
 60569 | checkpointer    |          | Timeout    | CheckpointWriteDelay   | 171242
 60573 | walwriter       |          | LWLock     | WALWrite               |  20021
 60683 | client backend  | pgbench  | Timeout    | SpinDelay              |      2
```

- категория `event_type='Activity'` используется, если фоновому процессу нечего делать
- событие `ClientRead` нормально, если оно не в открытой транзакции

Запросы к профилю `pg_wait_sampling`

- пример группировки для получения данных о том, каких событий больше всего ждали процессы экземпляра:

```
 select event_type, event, sum(count) count from pg_wait_sampling_profile where event_type<>'Activity'
group by event_type, event order by count desc;
  event_type |          event           | count
 ------------+--------------------------+--------
  IO         | WALSync                  |  84388
  LWLock     | WALWrite                 |   4549
```

- пример соединения с представлением `pg_stat_statements`:

```
 select calls, round(total_exec_time+total_plan_time) time, rows, shared_blks_hit hit, shared_blks_read
read, shared_blks_dirtied dirty, p.event_type ev_t, p.event, p.count, left(query,20) query from
pg_stat_statements join pg_wait_sampling_profile p using (queryid) order by p.count desc limit 7;
 calls | time   |  rows   |   hit    | read  | dirty |  ev_t  |     event      |count |     query
 ------+-------+--------+---------+-------+-------+--------+----------------+------+----------------
 415384|188566 | 415384 | 2572231 | 10815 | 27047 | IO     | DataFileRead   |15657 | UPDATE pgbench_accou
```

- пример соединения с представлением `pg_locks`:

```
 select p.pid, left(l.relation::regclass::text,19) relation, l.locktype, replace(l.mode,'Lock','') mode,
l.granted g, l.fastpath f, p.event_type type, p.event, p.count from pg_wait_sampling_profile p join
pg_locks l on p.pid = l.pid where event_type<>'Activity' and locktype<>'virtualxid' order by p.count
desc, p.pid, l.relation desc limit 30;
   pid  |      relation       |   locktype    |     mode      | g | f |  type  |   event    |  count
 -------+---------------------+---------------+---------------+---+---+--------+------------+---------
  11739 | pg_locks            | relation      | AccessShare   | t | t | Client | ClientRead | 6622131
  11731 |                     | transactionid | Exclusive     | t | f | IO     | WALSync    | 1906966
```

Сброс статистик

- перед выполнением сравнительных тестов бывает удобно сбросить статистики, чтобы накопленные с предыдущего теста данные не попадали в результат. Для этого используются функции сброса статистик:

```
select pg_stat_reset();
select pg_stat_reset_shared(null);
select pg_stat_reset_shared('bgwriter');
select pg_stat_reset_shared('archiver');
select pg_stat_reset_shared('io');
select pg_stat_reset_shared('wal');
select pg_stat_reset_shared('recovery_prefetch');
select pg_stat_reset_slru(null);
select pg_stat_statements_reset();
select pg_stat_kcache_reset();
select pg_wait_sampling_reset_profile();
select pg_qualstats_reset();
```

- для очистки кэша буферов остановить экземпляр: `pg_ctl stop -m fast`
- очистка pg_prewarm: `rm -f $PGDATA/autoprewarm.blocks`
- сброс на диск грязных страниц: `sync`
- очистка страничного кэша: `echo 3 > /proc/sys/vm/drop_caches`
- запуск экземпляра: `sudo systemctl restart tantor-se-server-16`

Практика

- Использование расширения pg_wait_sampling