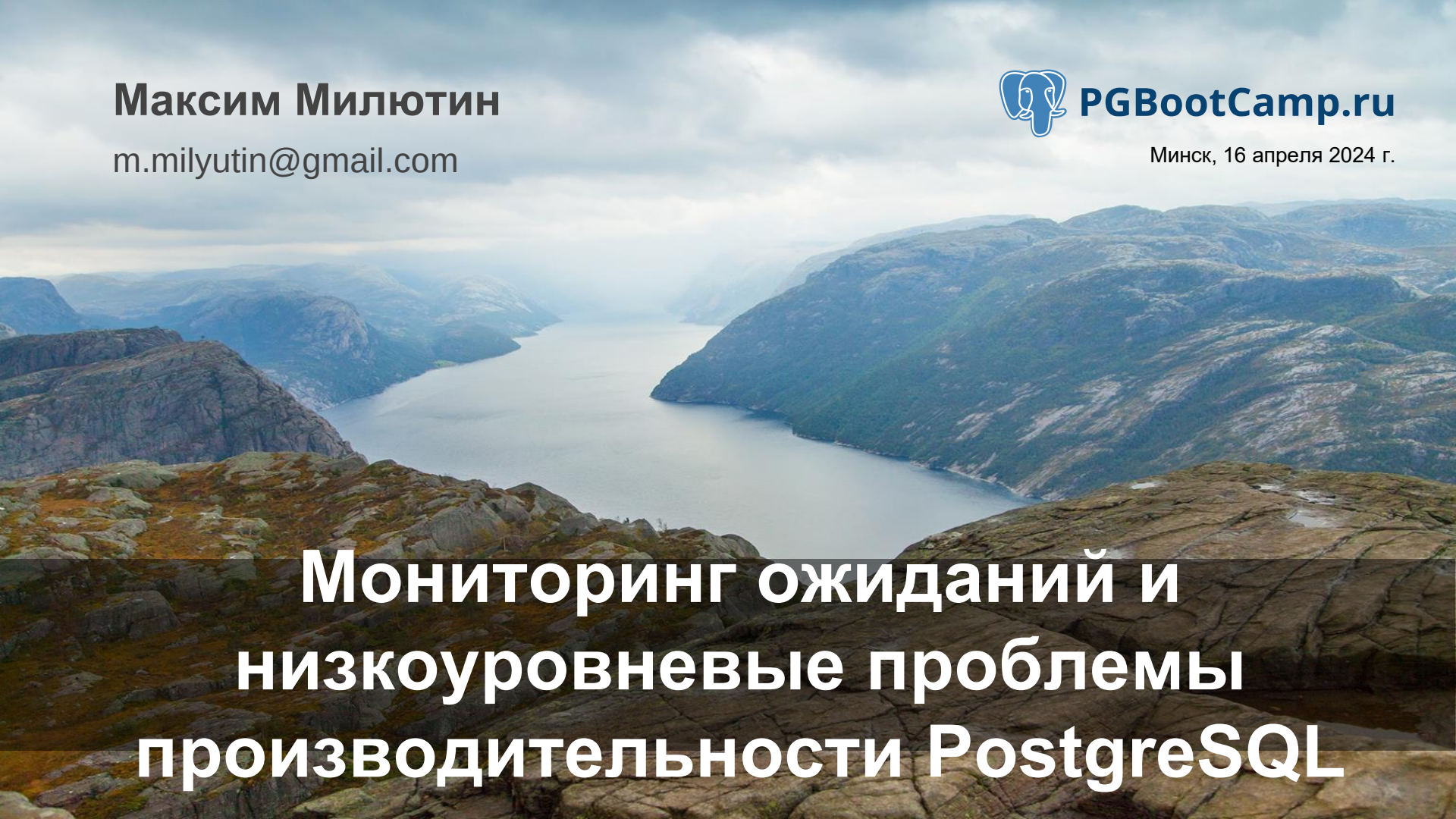
Максим Милютин

m.milyutin@gmail.com

PGBootCamp.ru

Минск, 16 апреля 2024 г.



Мониторинг ожиданий и низкоуровневые проблемы производительности PostgreSQL

О докладчике

- R&D SQL-движков для реляционных СУБД
  - улучшение планирования запросов методами ML
- Более 7 лет работы с базами данных и PostgreSQL
  - SQL прикладная разработка в e-commerce
  - DBA и консалтинг решений вокруг PostgreSQL
  - Разработка патчей к PostgreSQL и Greenplum и расширений к ним
  - R&D разработка вокруг OpenGauss/GaussDB

Понятие ожидания

- **Общая идея**: *база “ждёт”, когда не утилизируется CPU*
- Точнее - вне исполнения пользовательского СУБД специфичного кода
  - внутри системного блокирующего вызова (system CPU time)
  - в ожидании в очереди за ресурсом
  - внутри пользовательского внешнего неконтролируемого кода
- Общее время работы разбивается на сервисное рабочее время (CPU time) и время ожидания
  - из теории массового обслуживания: R = S + W

Метод R подход к мониторингу

- Метод R
  - выявить наиболее критичные (с точки зрения бизнеса) пользовательские операции
  - получить диагностические данные по ним для выявления причин задержек
  - оптимизировать те операции, которые дают наилучший совокупный эффект
- Мониторинг ожиданий - это инструмент для воплощения метода R
  - акцент на времени исполнения запроса и его составляющих
  - время выполнения запроса разбивается на CPU time (время обслуживания) и время ожиданий
- Разработан в Oracle как замена оценки системы через базовые health метрики
  - например, Buffer Cache Hit Ratio (BCHR) должно быть не меньше 90% (“*golden metric in database world”*)

Millsap, Cary V. Optimizing Oracle Performance. Sebastopol, CA: OReilly, 2003

Ретроспектива мониторинга ожиданий в Oracle

- *начало 90-х*: базовые метрики мониторинга
  - счётчики и средства взятие снапшота и их сравнения
  - логи по настраиваемым событиям
- *1992:* начиная с версии 7.0.12 логирование ожиданий внутри СУБД
  - как ответ на неспособность решить некоторые проблемы производительности базовыми метриками
  - представления V$SESSION_WAIT (позже V$SESSION) (аналог *pg_stat_activity*)
- *00-е*: широкое использование инструментария, включая *Active Session Sampling*

Ожидания в PostgreSQL

- *до 9.6* ожидания помечались битом в *pg_stat_activity*
- *версия 9.6* (2016 год):
  - **Heavy/Lightweight locks**, Buffer Pins
- *версия 10*:
  - латчи & сокеты, клиентские ожидания, ожидания внутри долгоиграющих циклов
  - **I/O**
  - для background и auxiliary процессов
- …
- *версия 17*:
  - кастомные ожидания для расширений (“Extension” тип)

Ожидания в PostgreSQL. Категории

- Lock:
  - транзакционные (heavy-weight) SQL локи
  - защита при конкурентном обновлении строки
- LWLock:
  - “мьютексы” для разделяемых структур структур внутри ядра
- IO:
  - дисковые чтения и запись
- Activity:
  - длительные “sleep” ожидания для background процессов
- IPC:
  - ожидания на других процессах

Ожидания в PostgreSQL

src/include/utils/wait_event.h

```
/* ----------
* pgstat_report_wait_start() -
*
*  Called from places where server process needs to wait.  This is called
*  to report wait event information.  The wait information is stored
*  as 4-bytes where first byte represents the wait event class (type of
*  wait, for different types of wait, refer WaitClass) and the next
*  3-bytes represent the actual wait event.  Currently 2-bytes are used
*  for wait event which is sufficient for current usage, 1-byte is
*  reserved for future usage.
*
* ----------
*/
static inline void
pgstat_report_wait_start(uint32 wait_event_info)
{
   /*
    * Since this is a four-byte field which is always read and written as
    * four-bytes, updates are atomic.
    */
   *(volatile uint32 *) my_wait_event_info = wait_event_info;
}
```

wait_event_info = classId | eventId

my_wait_event_info ~ MyProc->wait_event_info

Ожидания в PostgreSQL

src/backend/storage/file/fd.c

```
int
FileSync(File file, uint32 wait_event_info)
{
    ...

    pgstat_report_wait_start(wait_event_info);
    returnCode = pg_fsync(VfdCache[file].fd);
    pgstat_report_wait_end();

    ...
}
```

места ожидания обрамляется вызовами *pgstat_report_wait_start()* и *pgstat_report_wait_end()*

Мониторинг ожиданий

- Сэмплирование
  - кумулятивные счётчики по полям возможной группировки (*datname, usename, application_name, backend_type*, *query_id* , etc)
  - длительность ожидания ≈ кол-во сэмплов на период сэмплирования
  - точность / нагрузка на сервер регулируется частотой сэмплирования
- Трассировка
  - через подключение к внутренним prob`ам [1]
  - динамическая трассировка с помощью *ebpf*, etc.
  - значительный overhead при частых (~100К+) событиях ожиданий

1. https://www.postgresql.org/docs/16/dynamic-trace.html#TRACE-POINTS

Мониторинг через сэмплирование

- периодическое снятие снапшота с *pg_stat_activity*
  - дорого - берётся снапшот всего состояния бэкенда
  - приемлемый интервал сэмплирования **100 мс - 1 сек**
  - большой выбор смежных полей для группировки
- расширение *pg_wait_sampling*
  - сэмплирование `wait_event_info` флага внутри структур `PGPROC`
  - при интервале сэмплирования в **10 мс** overhead составляет единицы процента
  - варьирование только по `pid, query_id`
  - потенциально можно добавить другие поля

Мониторинг ожиданий через трассировку. OpenGauss

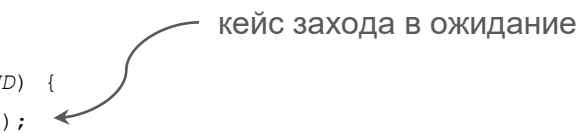
```
static inline void pgstat_report_waitevent(uint32 wait_event_info)
{
    PgBackendStatus* beentry = t_thrd.shemem_ptr_cxt.MyBEEntry;

    ...

    beentry->st_waitevent = wait_event_info;

    if (...enable_instr_track_wait && wait_event_info != WAIT_EVENT_END) {
        beentry->waitInfo.event_info.start_time = GetCurrentTimestamp();
    } else if (...enable_instr_track_wait && ... && wait_event_info == WAIT_EVENT_END) {
        TimestampTz current_time = GetCurrentTimestamp();
        int64 duration = current_time - beentry->waitInfo.event_info.start_time;
        UpdateWaitEventStat(&beentry->waitInfo, old_wait_event_info, duration, current_time);
        beentry->waitInfo.event_info.start_time = 0;
        beentry->waitInfo.event_info.duration = duration;
    }

    ...
}
```



Мониторинг ожиданий через трассировку. OpenGauss

```
void UpdateWaitEventStat(WaitInfo* instrWaitInfo, uint32 wait_event_info, int64 duration, TimestampTz currentTime)
{
   uint32 classId = wait_event_info & MASK_CLASS_ID;
   uint32 eventId = get_event_id(wait_event_info);
   ...
   switch (classId) {
             ...
       case PG_WAIT_LWLOCK:
           UpdateMinValue(duration,
               &(instrWaitInfo->event_info.lwlock_info[eventId].min_duration));
           instrWaitInfo->event_info.lwlock_info[eventId].counter++;
           instrWaitInfo->event_info.lwlock_info[eventId].total_duration += duration;
           UpdateMaxValue(duration, &(instrWaitInfo->event_info.lwlock_info[eventId].max_duration));
           instrWaitInfo->event_info.lwlock_info[eventId].last_updated = currentTime;
           break;
       ...
   }
}
```

для каждого ожидания по каждой сессии подсчитываются:

- кол-во ожиданий
- суммарная длительность
- статистика по длительности (min, max)

Мониторинг ожиданий через трассировку. OpenGauss

```
# select * from dbe_perf.wait_events order by total_wait_time desc limit 20;
     type     |              event               |   wait   | failed_wait | total_wait_time | avg_wait_time | max_wait_time | min_wait_time
--------------+----------------------------------+----------+-------------+-----------------+---------------+---------------+---------------
 STATUS       | wait cmd                         |    70852 |           0 |    265942030993 |       3753486 |  260796707550 |             2
 IO_EVENT     | CkptWaitPageWriterSync           |     4426 |           0 |       443212262 |        100138 |        103905 |        100058
 STATUS       | analyze                          |      256 |           0 |       172130457 |        672384 |       6273356 |          2465
 IO_EVENT     | BufHashTableSearch               | 56551675 |           0 |        56764090 |             1 |          1638 |             1
 STATUS       | HashAgg - build hash             |     1907 |           0 |        13454996 |          7055 |         35337 |             5
 IO_EVENT     | WALWrite                         |  2050189 |           0 |         6921869 |             3 |           350 |             3
 IO_EVENT     | DataFileRead                     |   598470 |           0 |         6680645 |            11 |          1889 |             3
 IO_EVENT     | unknown wait event               |     4426 |           0 |         1402726 |           316 |         44401 |           104
 IO_EVENT     | DoubleWriteFileWrite             |     5225 |           0 |         1177531 |           225 |         51284 |            96
 STATUS       | flush data                       |    87654 |           0 |         1160220 |            13 |         12499 |             1
 IO_EVENT     | ControlFileSyncUpdate            |     8854 |           0 |          708826 |            80 |         27991 |            44
 IO_EVENT     | buffer_strategy_get              |   617025 |           0 |          647851 |             1 |           227 |             1
 IO_EVENT     | DataFileWrite                    |   104384 |           0 |          571948 |             5 |           430 |             3
 IO_EVENT     | unknown wait event               |     4426 |           0 |          237129 |            53 |           197 |            25
 LWLOCK_EVENT | DoubleWriteLock                  |        3 |           0 |          197620 |         65873 |         80474 |         57409
 STATUS       | vacuum                           |     2943 |           0 |          156529 |            53 |           193 |            37
```

Мониторинг ожиданий через трассировку

- Возможно, имеет смысл сочетать с сэмплированием
- обязательно считать кол-во входов в ожидания
- трассировать по времени только “тяжёлые” и редкие в данный момент ожидания
  - временная характеристика будет в виде статистики по неполным данным

Расширение *pg_wait_sampling*

profile - это неограниченная хэш-таблица внутри background worker`а “*collector*” с счётчиками ожиданий для сессии (*pid*) и запроса (*query_id*)

```
# table pg_wait_sampling_profile order by count desc;
 pid    | event_type |        event        |       queryid        | count
--------+------------+---------------------+----------------------+-------
163263 | Activity   | AutoVacuumMain      |                    0 |  4503
163265 | Activity   | LogicalLauncherMain |                    0 |  4503
163262 | Activity   | WalWriterMain       |                    0 |  4503
163259 | Activity   | CheckpointerMain    |                    0 |  4503
163269 | Client     | ClientRead          |                    0 |  4503
163260 | Activity   | BgWriterHibernate   |                    0 |  4404
163260 | Activity   | BgWriterMain        |                    0 |    99
```

Расширение *pg_wait_sampling*. POC версия

profile - это ограниченная шаренная хэш-таблица с LRU вытеснением

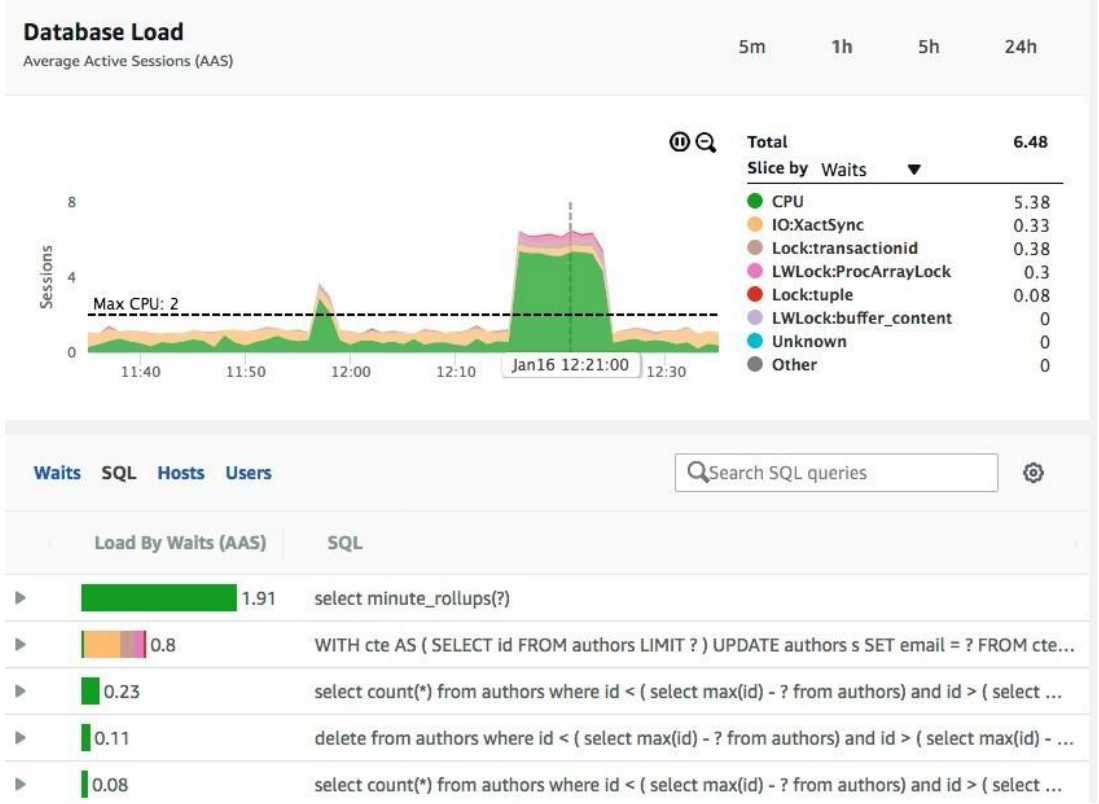
```
# table pg_wait_sampling_profile order by count desc;
 pid    | event_type |        event        |       queryid        | count
--------+------------+---------------------+----------------------+-------
(null)  | (null)     | (null)              |               (null) | 37069
163263  | Activity   | AutoVacuumMain      |                    0 |  4503
163342  | CPU time   | (null)              | -1426632887997485009 |  4503
163265  | Activity   | LogicalLauncherMain |                    0 |  4503
163262  | Activity   | WalWriterMain       |                    0 |  4503
163259  | Activity   | CheckpointerMain    |                    0 |  4503
163269  | Client     | ClientRead          |                    0 |  4503
163260  | Activity   | BgWriterHibernate   |                    0 |  4404
163260  | Activity   | BgWriterMain        |                    0 |    99
163504  | CPU time   | (null)              |                    0 |     1
```

Хак: count с null`ам - общий счётчик системы

в POC добавлены CPU time как состояние “без ожиданий”

запись с null`евым pid`ом - кумулятивный счётчик для вытестенных (или сброшенных) элементов из хэш-таблицы

Performance Insights. Database load борда



наложенные метрики ожидания, выраженные в AAS (*Average Active Sessions*) по временной шкале

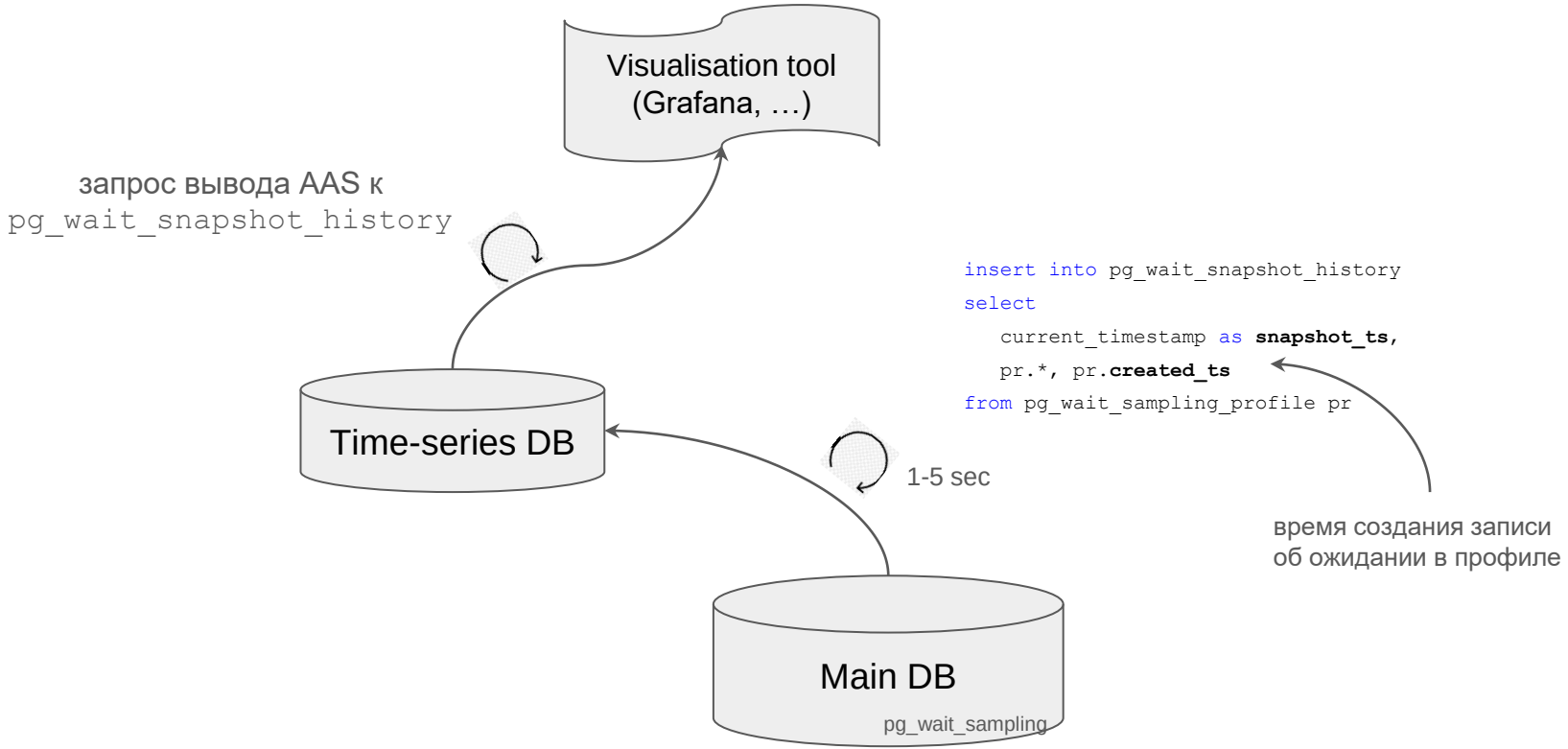
Топ SQL запросов с распределениями по ожиданиям по всему (?) временному интервалу

Average Active Sessions метрика

- AAS - количество сессий (процессов), активных в заданный квант времени
  - суммарное кол-во сэмплов по сессиям / кол-во проведённых сэмплов
- AAE (Average Active Execution) - аналогичная AAS величина, задаваемая отношением времен
  - суммарное время активности по всем сессиям / рассматриваемый временной интервал
- AAS и AAE соотносит count и time метрики
  - *pg_wait_sampling* & *pg_stat_kcache*: CPU time метрику по *query_id*
- соотношение AAS по CPU time к кол-ву ядер задаёт насыщение CPU
- Сумма активного времени в БД: $DBtime = \sum active\ sessions(t_i) * \Delta t$

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_PerfInsights.Overview.ActiveSessions.html

Общая архитектура системы мониторинга



Запрос вывода AAS. Sketch

```
with t as (
    select
        snapshot_ts,
        event, query_id, ...other grouping stuff,
        when created_ts = lead(created_ts, 1) over w then
            count - lead(count, 1)
        else
            count
        end as count_diff,
    from pg_wait_snapshot_history
    where
        created_ts between time_start and time_end
    window group_wnd as (partition by grouping stuff order by created desc)
)
select
    snapshot_ts,
    grouping stuff,
    sum(count_diff) / (cur_total_count - prev_total_count) as AAS
from t
group by
    snapshot_ts,
    grouping_stuff
```

разница count`ов со смежных снапшотов как count за период сэмплирования снапшотов

проверка на сброс элемента profile статистики

рассматриваемый интервал времени

суммарный count по всем сессиям на кол-во сэмплов между снапшотами

Желаемые фичи для мониторинга ожиданий в PostgreSQL

По мнению Jeremy Schneider (ссылка в конце)

- Wait event counters и Cumulative Times
- Аргументы (контекст) для ожидания (объект, номер блока и т.д.)
  - ожидание “SpinDelay” на долгом спинлоке :)
- “Честный” учёт CPU time (POSIX runusage)
  - расширение *pg_stat_kcache*: статистика по query_id
- Контекст для COMMIT/ROLLBACK команд
  - привязка к телу/id транзакции
- Детализация On-CPU состояния
  - стадия исполнения (parse/plan/execute/fetch)
  - текущий узел исполнения
- Улучшенная runtime visibility для PL конструкций

Низкоуровневые проблемные ожидания вокруг разделяемых компонентов

- Shared Buffer
  - *BufferMapping*
  - *BufferContent*
- массив структур PGPROC
  - *ProcArrayLock* ← GetSnapshotData()
- менеджер блокировок
  - *LockManager lwlock*
- WAL-буфер
  - *WALWriterLock*
- вспомогательные SLRU буфера
  - *SubtransSLRU lwlock*
- …

Низкоуровневые проблемные ожидания вокруг разделяемых компонентов. Deep dive.

Продолжение следует…

Источники и полезная литература

- Доклады Jeremy Schneider, Amazon RDS 🔥
  - https://youtu.be/q_sfdAFVIL8?t=8786
  - https://www.slideshare.net/ardentperf/wait-whats-going-on-inside-my-database-pass-2023-update
- Описание случаев && советы по разрешению ожиданий 🔥
  - https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/PostgreSQL.Tuning.concepts.summary.html

PGBootCamp.ru

Спасибо за внимание!
Вопросы. Критика. Пожелания…

Максим Милютин milyutinma@gmail.com