Оптимизация OLTP-нагрузки

Сергей Новиков

ЕДИНЫЙ ЦУПИС

PGConf.Russia
2024

Кто мы

ЕДИНЫЙ ЦУПИС – высоконагруженная fintech-экосистема, одна из крупнейших на рынке.

Собственный электронный кошелёк.

Миллионы платежей ежедневно.

SLA свыше 99,99%.

Чего в докладе не будет

- Зубодробительные EXPLAIN
- Тюнинг железа / операционной системы
- Rocket Science 4 Rock Stars

Мониторинг

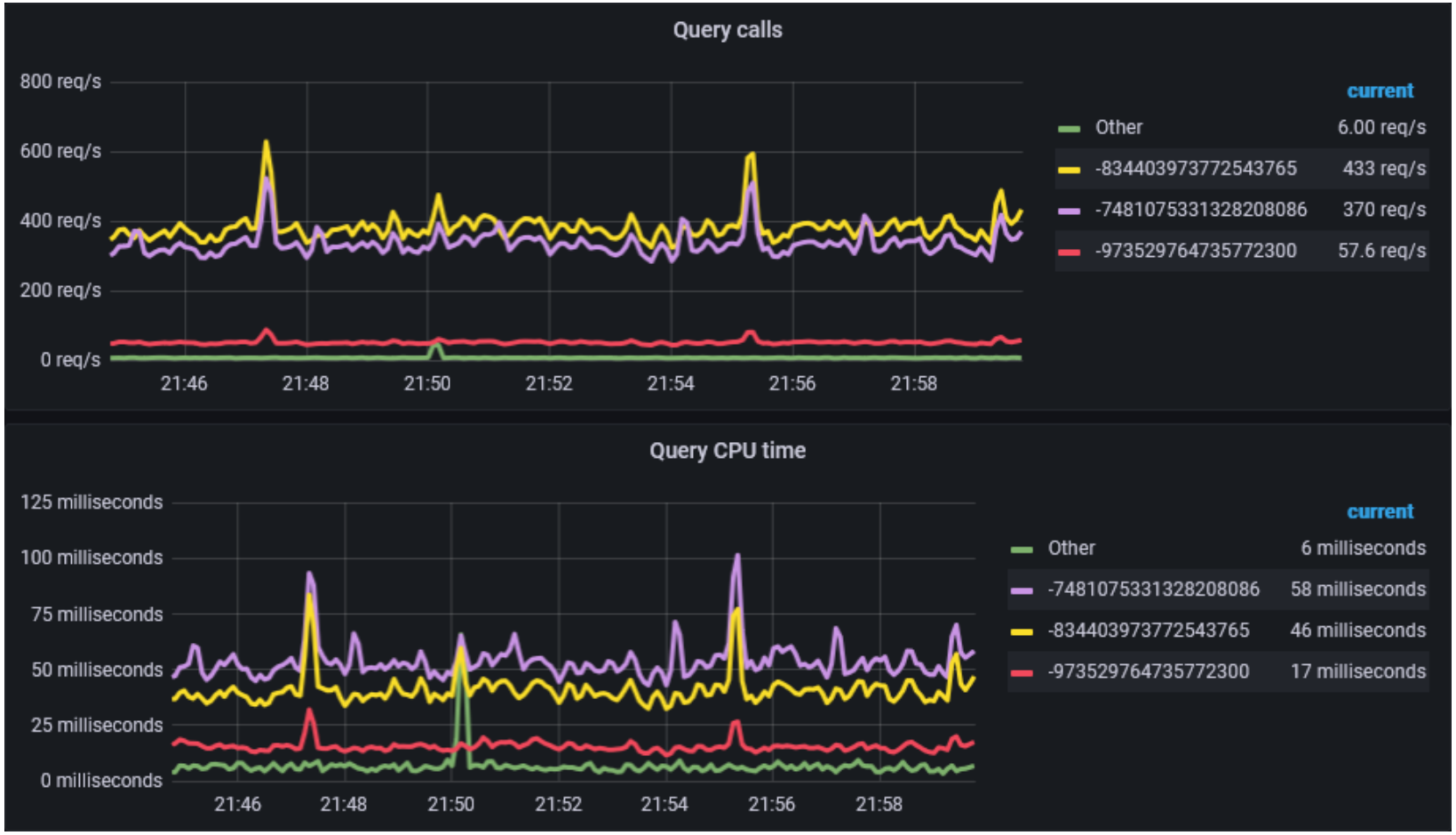
Для мониторинга запросов в реальном времени нужны:

- pg_stat_statements
- postgres_exporter с кастомными запросами
- VictoriaMetrics

Мониторинг

```
-- Запрос для снятия метрик.
SELECT
    current_database() AS database,
    queryid AS query_id,
    sum(calls) AS calls,
    sum(total_exec_time) AS total_time,
    sum(blk_read_time + blk_write_time) AS io_time
FROM
    pg_stat_statements(FALSE)
GROUP BY
    queryid;
```

Мониторинг


Query calls
800 req/s
600 req/s
400 req/s
200 req/s
0 req/s
21:46
21:48
21:50
21:52
21:54
21:56
21:58
current
Other 6.00 req/s
-834403973772543765 433 req/s
-7481075331328208086 370 req/s
-973529764735772300 57.6 req/s
Query CPU time
125 milliseconds
100 milliseconds
75 milliseconds
50 milliseconds
25 milliseconds
0 milliseconds
21:46
21:48
21:50
21:52
21:54
21:56
21:58
current
Other 6 milliseconds
-7481075331328208086 58 milliseconds
-834403973772543765 46 milliseconds
-973529764735772300 17 milliseconds

Мониторинг


Query total time
150 milliseconds
100 milliseconds
50 milliseconds
0 milliseconds
21:46
21:48
21:50
21:52
21:54
21:56
21:58
current
Other 6 milliseconds
-7481075331328208086 81 milliseconds
-834403973772543765 58 milliseconds
-973529764735772300 46 milliseconds
Query I/O time
60 milliseconds
50 milliseconds
40 milliseconds
30 milliseconds
20 milliseconds
10 milliseconds
0 milliseconds
21:46
21:48
21:50
21:52
21:54
21:56
21:58
current
Other 0 milliseconds
-7481075331328208086 23 milliseconds
-973529764735772300 29 milliseconds
-834403973772543765 11 milliseconds

Мониторинг

CPU – высокое потребление user time:

- Seq scan
- JIT

Мониторинг

CPU – высокое потребление system time:

- Параллелизм
- Короткоживущие коннекты

Оптимизация с помощью n_distinct

Хрестоматийный пример: выборка последних N событий по критерию. Например, последние 10 платежей клиента из общей таблицы платежей.

```
SELECT * FROM {table} WHERE client_id = 1337 ORDER BY id DESC LIMIT 10;
```

Оптимизация с помощью n_distinct

В общем случае, всё работает нормально, но для некоторых клиентов иногда план кардинально меняется на backward index scan по полю сортировки.

```
Index Scan Backward using {primary_key} on {table}
(cost=0.43..481639.64 rows=1020 width=135)
(actual time=14157.206..14157.206 rows=10 loops=1)
Filter: (client_id = 1337)
Rows Removed by Filter: 11419635
```

Оптимизация с помощью n_distinct

Возможные решения:

- Поменять поле для сортировки на неиндексированное

Оптимизация с помощью n_distinct

Возможные решения:

- Поменять поле для сортировки на неиндексированное
- Сортировать по выражению, например, по id+0

Оптимизация с помощью n_distinct

Возможные решения:

- Поменять поле для сортировки на неиндексированное
- Сортировать по выражению, например, по id+0
- Удалить проблемный индекс или построить составной

Оптимизация с помощью n_distinct

Возможные решения:

- Поменять поле для сортировки на неиндексированное
- Сортировать по выражению, например, по id+0
- Удалить проблемный индекс или построить составной
- Настроить селективность для критерия поиска

Оптимизация с помощью n_distinct

Часто проблемы с запросами вызваны неправильной оценкой кардинальности.

Когда кардинальность недооценена, мы получаем вечный nested loop (типично для OLAP).

Вариант решения: Перепланирование в реальном времени.

Оптимизация с помощью n_distinct

Но если кардинальность переоценена, мы также можем выбрать неоптимальный план, что особенно влияет на OLTP.

Чаще всего проявляется, как выбор планировщиком не того индекса. В том числе выбор разных индексов в секциях однородной секционированной таблицы.

Оптимизация с помощью n_distinct

```
SELECT
    n_distinct
FROM
    pg_stats
WHERE
    tablename = '{table}'
    AND
    attname = 'client_id';

 n_distinct
------------
     370754
```

Оптимизация с помощью n_distinct

Причины некорректной оценки селективности:

- Очень большая таблица
- Несовершенство ANALYZE
- default_statistics_target по умолчанию 100

Оптимизация с помощью n_distinct

Причины некорректной оценки селективности:

- Очень большая таблица
- Несовершенство ANALYZE
- ~~default_statistics_target по умолчанию 100~~
  Пожалуйста, не трогайте его…

Оптимизация с помощью n_distinct

Возможные значения n_distinct:

- Положительные целые – количество (enum)
- Отрицательные – доля уникальных от общего числа. Например, для уникальных ключей n_distinct всегда -1

Оптимизация с помощью n_distinct

```
-- Уникальных значений — почти 5% от общего числа строк.
SELECT count(*) AS all, count(DISTINCT client_id) FROM {table};
    all    |  count
-----------+---------
 178263417 | 8474965
```

Оптимизация с помощью n_distinct

```
-- Ручная фиксация n_distinct.
ALTER TABLE {table} ALTER COLUMN client_id SET (n_distinct = -0.05);

ANALYZE {table};
```

Оптимизация с помощью n_distinct

Дополнительные замечания:

- Вариант настройки n_distinct_inherited не работает с секционированными таблицами
- Значение, зафиксированное вручную, само уже не поменяется – лучше периодически перепроверять
- Корректная настройка селективности очень часто решает и проблему переоценки стоимости

Оптимизация отключением оптимизаций

Переоценка стоимости вызывает оптимизации для OLAP:

- JIT (начиная со 100000)
- Параллелизм (начиная с 1000)

Оптимизация отключением оптимизаций

Переоценка стоимости вызывает оптимизации для OLAP:

- JIT (начиная со 100000)
- Параллелизм (начиная с 1000)

Особенно “любит” портить стоимость поиск по JSON…

Оптимизация отключением оптимизаций

Не все prepared statements одинаково полезны:

- В короткоживущих коннектах пользы от них нет

Оптимизация отключением оптимизаций

Не все prepared statements одинаково полезны:

- В короткоживущих коннектах пользы от них нет
- Закэшироваться может неудачный план

Оптимизация отключением оптимизаций

Не все prepared statements одинаково полезны:

- В короткоживущих коннектах пользы от них нет
- Закэшироваться может неудачный план
- Мешают использовать pgBouncer

Оптимизация индексов

Не все индексы одинаково полезны:

- Для неравномерных значений лучше использовать условные индексы (и не на сам критерий)

Оптимизация индексов

Не все индексы одинаково полезны:

- Для неравномерных значений лучше использовать условные индексы (и не на сам критерий)
- Больше индексов – больше вариантов планировщику выстрелить вам в ногу

Оптимизация индексов

Не все индексы одинаково полезны:

- Для неравномерных значений лучше использовать условные индексы (и не на сам критерий)
- Больше индексов – больше вариантов планировщику выстрелить вам в ногу
- Добившись index-only scan, не останавливайте поиски

Бывает и такое – на реплике медленнее

Симптомы:

- На мастере запрос выполняется быстро
- На реплике гораздо медленнее
- Проблема воспроизводится на всех репликах

Бывает и такое – на реплике медленнее

Симптомы:

- На мастере запрос выполняется быстро
- На реплике гораздо медленнее
- Проблема воспроизводится на всех репликах
- Планы запроса везде одинаковые

Бывает и такое – на реплике медленнее

Симптомы:

- На мастере запрос выполняется быстро
- На реплике гораздо медленнее
- Проблема воспроизводится на всех репликах
- Планы запроса везде одинаковые
- А вот объём чтения разный

Бывает и такое – на реплике медленнее

```
-- МАСТЕР:
Index Scan using {index} on {table} (cost=0.56..2.58 rows=1
width=366) (actual time=2.222..2.222 rows=0 loops=348)
Index Cond: ({column} IS NULL)
Buffers: shared hit=453096

-- РЕПЛИКА:
Index Scan using {index} on {table} (cost=0.56..2.58 rows=1
width=366) (actual time=54.875..54.875 rows=0 loops=348)
Index Cond: ({column} IS NULL)
Buffers: shared hit=14117664
```

Бывает и такое – на реплике медленнее

Корневая причина – на репликах игнорируются hint bits.

Autovacuum не чистил проблемную таблицу три дня.

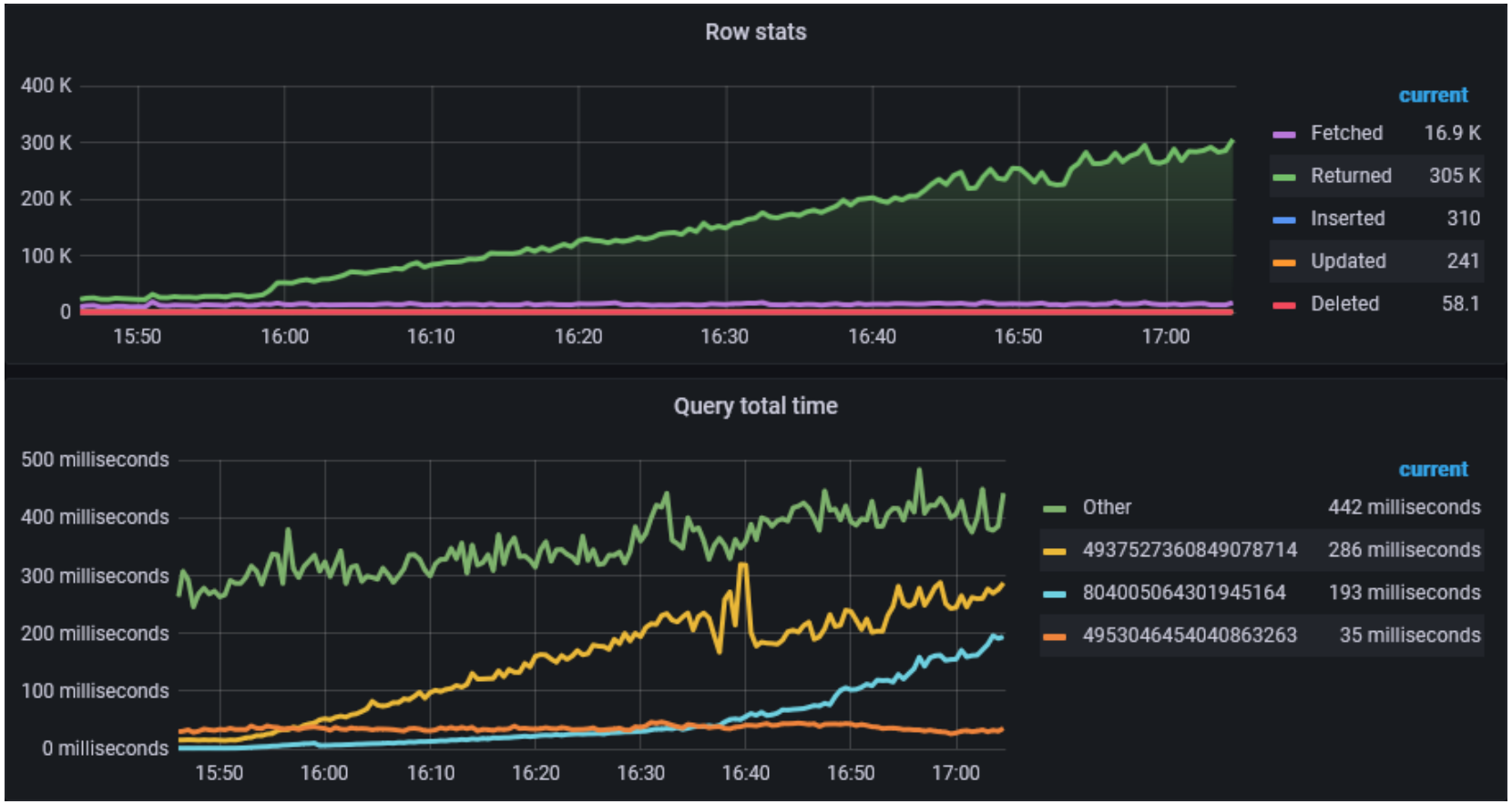
Решение – настроить регулярный autovacuum.

Бывает и такое – логический слот-убийца

Симптомы:

- Когда появляется неактивный логический слот, начинается деградация
- Деградировать могут самые разные запросы
- Больше всего страдают активно обновляемые таблицы

Бывает и такое – логический слот-убийца



Бывает и такое – логический слот-убийца

Корневая причина – логический слот резервирует catalog_xmin, что блокирует автоочистку системного каталога.

В системах с частым и агрессивным autoanalyze начинает распухать pg_statistic.

В результате деградирует **планирование** запросов.

Решение – либо удалять неактивные логические слоты, либо пересмотреть частоту срабатывания autoanalyze.

Бывает и такое – циферки не сходятся

Симптомы:

- При нагрузочном тестировании база выдерживает 200 RPS, хотя график транзакций показывает 20k TPS
- Запрос выполняется 1,5 мс, по расчётам 30 коннектов должны выдерживать 20k RPS

Бывает и такое – циферки не сходятся

Симптомы:

- При нагрузочном тестировании база выдерживает 200 RPS, хотя график транзакций показывает 20k TPS
- Запрос выполняется 1,5 мс, по расчётам 30 коннектов должны выдерживать 20k RPS
- Приложение считает, что запрос выполняется 150 мс

Бывает и такое – циферки не сходятся

Корневая причина – Spring не знает типы параметров в запросе и решает спросить их у базы данных.

Тип каждого параметра запрашивается отдельным служебным вызовом. В запросе ровно 100 параметров.

Решение – напрямую указать типы или подтюнить Spring.

Спасибо за внимание!

Сергей Новиков

ЕДИНЫЙ ЦУПИС