Оптимизация запросов в PostgreSQL - live at PG Day

Алексей Ермаков
Илья Космодемьянский

PostgreSQL-Consulting.com

Постановка задачи

- Есть postgresql, которому плохо
- Куда смотреть
- Что делать
- В каком порядке
- Кто виноват

План действий

- Проверка конфигурации на предмет глупых ошибок
- Диагностика проблем со стороны системы
- Диагностика проблем со стороны базы

Настройки базы и ОС

- shared_buffers
- autovacuum
- checkpoint_segments
- barrier

Диагностика проблем со стороны системы

- top
- iotop
- iostat
- ping

Диагностика проблем со стороны базы

- pg_stat_activity
- log analyze tools: pgfouine, pgbadger, loganalyzer
- pg_stat_statements

pg_stat_statements

```
total time: 50:49:48 (IO: 0.64%)
total queries: 301,163,398 (unique: 9,206)
report for all databases, version 0.9.3 @ PostgreSQL 9.2.13
tracking top 10000 queries, logging 100ms+ queries

================================================================================================================
pos:1  total time: 14:39:43 (28.8%, CPU: 28.8%, IO: 36.8%) calls: 4,895,890 (1.63%)
avg_time: 10.78ms (IO: 0.8%)
user: bravo  db: echo  rows: 4,895,890  query:
SELECT sum(o.golf) as golf, sum(o.romeo) as romeo, sum(o.whiskey) as whiskey,
               sum(o.hotel) as hotel FROM oscar AS o LEFT JOIN uniform AS u ON u.kilo = o.kilo JOIN
```

pg_stat_statements

```
pos:2  total time: 09:54:58 (19.5%, CPU: 19.5%, IO: 17.0%)  calls: 87,527,549 (29.06%)
avg_time: 0.41ms (IO: 0.0%)
user: all  db: all  rows: 940,686,425  query:
other

================================================================================
pos:3  total time: 07:34:05 (14.9%, CPU: 15.0%, IO: 0.0%)  calls: 5,062 (0.00%)
avg_time: 5382.34ms (IO: 0.0%)
user: bravo  db: echo  rows: 45,967  query:
SELECT * FROM users WHERE params LIKE ? AND user_id != ? LIMIT ?
```

demo: query #1

```
SELECT b.id, b.rating
FROM books b
LEFT JOIN complaints com ON (com.book_id = b.id AND com.user_id = 155473)
WHERE b.is_deleted IS FALSE AND b.type_id != 4 AND com.is_hide IS NOT TRUE
ORDER BY b.rating DESC LIMIT 26;
```

demo: query #1

```
SELECT b.id, b.rating
FROM books b
WHERE b.is_deleted IS FALSE AND b.type_id != 4
AND NOT EXISTS (SELECT 1 FROM complaints com where
com.book_id = b.id and com.user_id = 155473 and is_hide = true)
ORDER BY b.rating DESC LIMIT 26;
```

demo: query #2

```
select * from cities where combined ilike '%viborg%' order by population desc limit 3;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Limit  (cost=34848.73..34848.73 rows=3 width=48)
 (actual time=944.693..944.693 rows=3 loops=1)
   ->  Sort  (cost=34848.73..34849.11 rows=152 width=48)
             (actual time=944.693..944.693 rows=3 loops=1)
         Sort Key: population
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on cities  (cost=0.00..34846.76 rows=152 width=48)
                                 (actual time=664.966..944.672 rows=5 loops=1)
               Filter: (combined ~~* '%viborg%'::text)
               Rows Removed by Filter: 1533576
 Planning time: 0.691 ms
 Execution time: 944.722 ms
```

demo: query #2

```
pgday=# create extension pg_trgm;
pgday=# create index concurrently cities_combined_trgm on cities using gin(combined gin_trgm_ops);
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Limit  (cost=624.14..624.15 rows=3 width=48) (actual time=0.558..0.558 rows=3 loops=1)
   ->  Sort  (cost=624.14..624.52 rows=152 width=48) (actual time=0.557..0.557 rows=3 loops=1)
         Sort Key: population
         Sort Method: quicksort  Memory: 25kB
         ->  Bitmap Heap Scan on cities  (cost=57.18..622.18 rows=152 width=48) (actual time=0.523..0
               Recheck Cond: (combined ~~* '%viborg%'::text)
               Heap Blocks: exact=4
               ->  Bitmap Index Scan on cities_combined_idx  (cost=0.00..57.14 rows=152 width=0) (act
                     Index Cond: (combined ~~* '%viborg%'::text)
 Planning time: 0.865 ms
 Execution time: 0.619 ms
```

demo: query #3

```
select distinct country_code from cities;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate  (cost=34846.76..34847.82 rows=106 width=3)
                (actual time=396.764..396.772 rows=121 loops=1)
   Group Key: country_code
   ->  Seq Scan on cities  (cost=0.00..31012.81 rows=1533581 width=3)
                           (actual time=0.012..160.341 rows=1533581 loops=1)
 Planning time: 0.137 ms
 Execution time: 396.838 ms
```

demo: query #3

Loose index scan

```
WITH RECURSIVE t AS (
(SELECT min(country_code) AS country_code FROM cities)
UNION ALL
SELECT (SELECT min(country_code) FROM cities WHERE country_code > t.country_code) AS country_code FRO
)
SELECT country_code FROM t WHERE country_code IS NOT NULL;
```

demo: query #4

```
select * FROM bravo
WHERE (bravo.sierra >= 9) AND (tango_id IS NULL OR tango_id IN(0,1))
AND (bravo.echo_count + bravo.delta_count + bravo.hotel_count + bravo.oscar_count >= 1)
AND bravo.is_visible = 0 AND bravo.yankee_id = 1
ORDER BY bravo.delta_count desc, bravo.created_at asc LIMIT 10;

 Limit  (cost=44913.42..44913.45 rows=10 width=72) (actual time=269.652..269.654 rows=10 loops=1)
   ->  Sort  (cost=44913.42..45166.69 rows=101305 width=72) (actual time=269.651..269.653 rows=10 loo
         Sort Key: delta_count, created_at
         Sort Method: top-N heapsort  Memory: 35kB
         ->  Seq Scan on bravo  (cost=0.00..42724.26 rows=101305 width=72) (actual time=0.015..212.14
               Filter: ((sierra >= 9) AND ((tango_id IS NULL) OR (tango_id = ANY ('{0,1}'::integer[])
               Rows Removed by Filter: 95131
 Planning time: 0.322 ms
 Execution time: 269.701 ms
```

demo: query #4

```
create index concurrently on bravo using btree(delta_count desc, created_at)
where (bravo.echo_count + bravo.delta_count + bravo.hotel_count + bravo.oscar_count >= 1);
```

pg_stat_statements

- sql/global_reports/query_stat_total.sql
- `https://github.com/PostgreSQL-Consulting/pg-utils`

Полезные ссылки

- depesz: Explaining the unexplainable
- `https://wiki.postgresql.org/wiki/Loose_indexscan`
- `https://github.com/PostgreSQL-Consulting/pg-utils`
- `http://blog.postgresql-consulting.com/`

Вопросы?

- ik@postgresql-consulting.com
- alexey.ermakov@postgresql-consulting.com