

Отладка

- [Отладочные запросы](#)
- [Выгрузка результатов запросов](#)
- [Простое обнаружение проблем производительности в PostgreSQL](#)
- [Слоты репликации](#)

Отладочные запросы

Вывод списка активных запросов и его длительность

```
SELECT
    pid,
    datname,
    username,
    client_addr,
    xact_start,
    query_start,
    (state_change-query_start) AS query_duration,
    wait_event_type,
    wait_event,
    state,
    query
FROM
    pg_stat_activity
WHERE
    state != 'idle'
AND
    username NOT IN ('postgres','replicator','zabbix');
```

20 самых нагруженных запросов

```
SELECT
    substring(query, 1, 50) AS short_query,
    query,
    round(total_time::numeric, 2) AS total_time,
```

```
calls,  
round(mean_time::numeric, 2) AS mean,  
round((100 * total_time / sum(total_time::numeric) OVER ()):numeric, 2) AS percentage_cpu  
FROM  
pg_stat_statements  
ORDER BY  
total_time DESC LIMIT 20;
```

Общее количество активных и оставшихся соединений

```
SELECT  
max_conn,  
used,  
res_for_super,  
max_conn-used-res_for_super res_for_normal  
FROM  
(SELECT count(*) used FROM pg_stat_activity) t1,  
(SELECT setting::int res_for_super FROM pg_settings WHERE name=$$superuser_reserved_connections$$) t2,  
(SELECT setting::int max_conn FROM pg_settings WHERE name=$$max_connections$$) t3;
```

Количество подключений в разрезе баз данных и адресов клиентов

```
SELECT count(*), client_addr, datname FROM pg_stat_activity GROUP BY client_addr, datname;
```

Количество подключений к базе данных в разрезе адресов

КЛИЕНТОВ

```
SELECT count(*), client_addr, datname FROM pg_stat_activity WHERE datname = 'db_name' GROUP BY  
client_addr, datname;
```

Выгрузка результатов запросов

В html

```
$ psql -h <host> -p <port> -U postgres -H -c "SQL command"
```

Простое обнаружение проблем производительности в PostgreSQL

Существует ли в мире очень большая и крупная база данных, которая время от времени не страдает от проблем с производительностью? Держу пари, что их не так уж много. Поэтому каждый DBA (администратор базы данных), отвечающий за PostgreSQL, должен знать, как отслеживать потенциальные проблемы производительности, чтобы выяснить, что на самом деле происходит.

Повышение производительности PostgreSQL после настройки параметров

Многие думают, что изменение параметров в `postgresql.conf` — это реальный путь к успеху. Однако это не всегда так. Конечно, чаще всего хорошие параметры конфигурации базы данных очень полезны. Тем не менее, во многих случаях реальные проблемы будут возникать из-за странного запроса, скрытого глубоко в некоторой логике приложения. Даже вполне вероятно, что запросы, вызывающие реальные проблемы, не являются теми, на которые вы обратили внимание. Возникает естественный вопрос: как мы можем отследить эти запросы и выяснить, что на самом деле происходит? Мой любимый инструмент для этого — `pg_stat_statements`, который всегда должен быть включен по моему мнению, если вы используете PostgreSQL 9.2 или выше (пожалуйста, не используйте его в более старых версиях).

Включение `pg_stat_statements`

Чтобы включить `pg_stat_statements` на вашем сервере, измените следующую строку в `postgresql.conf` и перезапустите PostgreSQL:

```
shared_preload_libraries = 'pg_stat_statements'
```

После загрузки этого модуля на сервер PostgreSQL автоматически начнет собирать информацию. Хорошо то, что накладные расходы модуля действительно очень низкие (накладные расходы в основном просто «шум»).

Затем выполните следующую команду для создания представления, необходимого для доступа к данным:

```
CREATE EXTENSION pg_stat_statements;
```

Прелесть здесь в том, что тип запроса, который наиболее трудоемкий, естественно будет отображаться в верхней части списка. Лучший способ — пройти от первого до, скажем, 10-го запроса и посмотреть, что там происходит.

По моему мнению, невозможно настроить систему без просмотра наиболее трудоемких запросов на сервере базы данных.

Углубленный анализ производительности PostgreSQL

`pg_stat_statements` может предложить гораздо больше, чем просто запрос и время, которое он занял. Вот структура представления:

test=# \d pg_stat_statements

View "public.pg_stat_statements"

Column	Type	Collation	Nullable	Default
userid	oid			
dbid	oid			
queryid	bigint			
query	text			
calls	bigint			
total_time	double precision			
min_time	double precision			
max_time	double precision			
mean_time	double precision			
stddev_time	double precision			
rows	bigint			
shared_blks_hit	bigint			
shared_blks_read	bigint			
shared_blks_dirtied	bigint			
shared_blks_written	bigint			
local_blks_hit	bigint			
local_blks_read	bigint			
local_blks_dirtied	bigint			
local_blks_written	bigint			

temp_blks_read	bigint			
temp_blks_written	bigint			
blk_read_time	double precision			
blk_write_time	double precision			

Вполне полезно посмотреть и на столбец `stddev_time`. Если стандартное отклонение велико, можно ожидать, что некоторые из этих запросов будут быстрыми, а некоторые — медленными, что может привести к ухудшению работы пользователей.

Столбец `rows` также может быть достаточно информативным. Предположим, что 1000 вызовов вернули 1.000.000.000 строк: фактически это означает, что каждый вызов в среднем возвращал 1 миллион строк. Легко понять, что возвращать столько данных все время не слишком хорошо.

Если требуется проверить, показывает ли определенный тип запроса плохую производительность при кэшировании, будет интересен `shared_*`. Вкратце: PostgreSQL может сообщить вам частоту обращений к кешу для каждого отдельного типа запроса в случае, если включен `pg_stat_statements`.

Также имеет смысл взглянуть на поля `temp_blks_*`. Каждый раз, когда PostgreSQL должен обратиться к диску для сортировки или материализации, потребуются временные блоки.

Наконец-то есть `blk_read_time` и `blk_write_time`. Обычно эти поля пусты, если не включен `track_io_timing`. Идея здесь заключается в том, чтобы иметь возможность измерять количество времени, которое определенный тип запроса тратит на ввод-вывод. Это позволит вам ответить на вопрос, привязана ли ваша система к вводу/выводу или к ЦП. В большинстве случаев рекомендуется включить **I/O timing**, поскольку это даст вам важную информацию.

Работа с Java и Hibernate

`pg_stat_statements` дает хорошую информацию. Однако в некоторых случаях запрос может быть прерван из-за переменной конфигурации:

```
test=# SHOW track_activity_query_size;
track_activity_query_size
-----
1024
(1 row)
```

Для большинства приложений 1024 байта абсолютно достаточно. Однако обычно это не тот случай, если вы используете Hibernate или Java. Hibernate имеет тенденцию посылать безумно длинные запросы к базе данных, и поэтому код SQL может быть обрезан задолго до запуска соответствующих частей (например, предложение FROM и т.д.). Поэтому имеет смысл увеличить `track_activity_query_size` до более высокого значения (возможно 32.786).

Полезные запросы для выявления узких мест в PostgreSQL

Есть один запрос, который я нашел особенно полезным в этом контексте: Следующий запрос показывает 20 запросов, занимающих много времени:

```
test=# SELECT substring(query, 1, 50) AS short_query,
       round(total_time::numeric, 2) AS total_time,
       calls,
       round(mean_time::numeric, 2) AS mean,
       round((100 * total_time / sum(total_time::numeric) OVER ()):numeric, 2) AS percentage_cpu
FROM   pg_stat_statements
ORDER BY total_time DESC
LIMIT 20;
```

short_query	total_time	calls	mean	percentage_cpu
SELECT name FROM (SELECT pg_catalog.lower(name) A	11.85	7	1.69	38.63
DROP SCHEMA IF EXISTS performance_check CASCADE;	4.49	4	1.12	14.64

```

CREATE OR REPLACE FUNCTION performance_check.pg_st | 2.23 | 4 | 0.56 | 7.27
SELECT pg_catalog.quote_ident(c.relname) FROM pg_c | 1.78 | 2 | 0.89 | 5.81
SELECT a.attname, + | 1.28 | 1 | 1.28 | 4.18
SELECT substring(query, ?, ?) AS short_query, roun | 1.18 | 3 | 0.39 | 3.86
CREATE OR REPLACE FUNCTION performance_check.pg_st | 1.17 | 4 | 0.29 | 3.81
SELECT query FROM pg_stat_activity LIMIT ?; | 1.17 | 2 | 0.59 | 3.82
CREATE SCHEMA performance_check; | 1.01 | 4 | 0.25 | 3.30
SELECT pg_catalog.quote_ident(c.relname) FROM pg_c | 0.92 | 2 | 0.46 | 3.00
SELECT query FROM performance_check.pg_stat_activi | 0.74 | 1 | 0.74 | 2.43
SELECT * FROM pg_stat_statements ORDER BY total_ti | 0.56 | 1 | 0.56 | 1.82
SELECT query FROM pg_stat_statements LIMIT ?; | 0.45 | 4 | 0.11 | 1.45
GRANT EXECUTE ON FUNCTION performance_check.pg_sta | 0.35 | 4 | 0.09 | 1.13
SELECT query FROM performance_check.pg_stat_statem | 0.30 | 1 | 0.30 | 0.96
SELECT query FROM performance_check.pg_stat_activi | 0.22 | 1 | 0.22 | 0.72
GRANT ALL ON SCHEMA performance_check TO schoenig_ | 0.20 | 3 | 0.07 | 0.66
SELECT query FROM performance_check.pg_stat_statem | 0.20 | 1 | 0.20 | 0.67
GRANT EXECUTE ON FUNCTION performance_check.pg_sta | 0.19 | 4 | 0.05 | 0.62
SELECT query FROM performance_check.pg_stat_statem | 0.17 | 1 | 0.17 | 0.56
(20 rows)

```

Последний столбец особенно примечателен: он показывает процент общего времени, потраченного на один запрос. Это поможет вам выяснить, насколько влияет запрос на общую производительность или не влияет.

Дополнительные материалы

- [Обнаружение медленных запросов](#)

Слоты репликации

Вывести список слотов репликации

```
SELECT * FROM pg_replication_slots;
```

Создать слот репликации

```
SELECT pg_create_physical_replication_slot('slot_name');
```

Удалить слот репликации

```
SELECT pg_drop_replication_slot('slot_name')
```

Список литературы

- [Функции для системного администрирования](#)