Анализ исторической нагрузки PostgreSQL

Расширение *pg_profile*

PGConf.Russia 2020

Андрей Зубков (DBA)
zubkov@moonset.ru

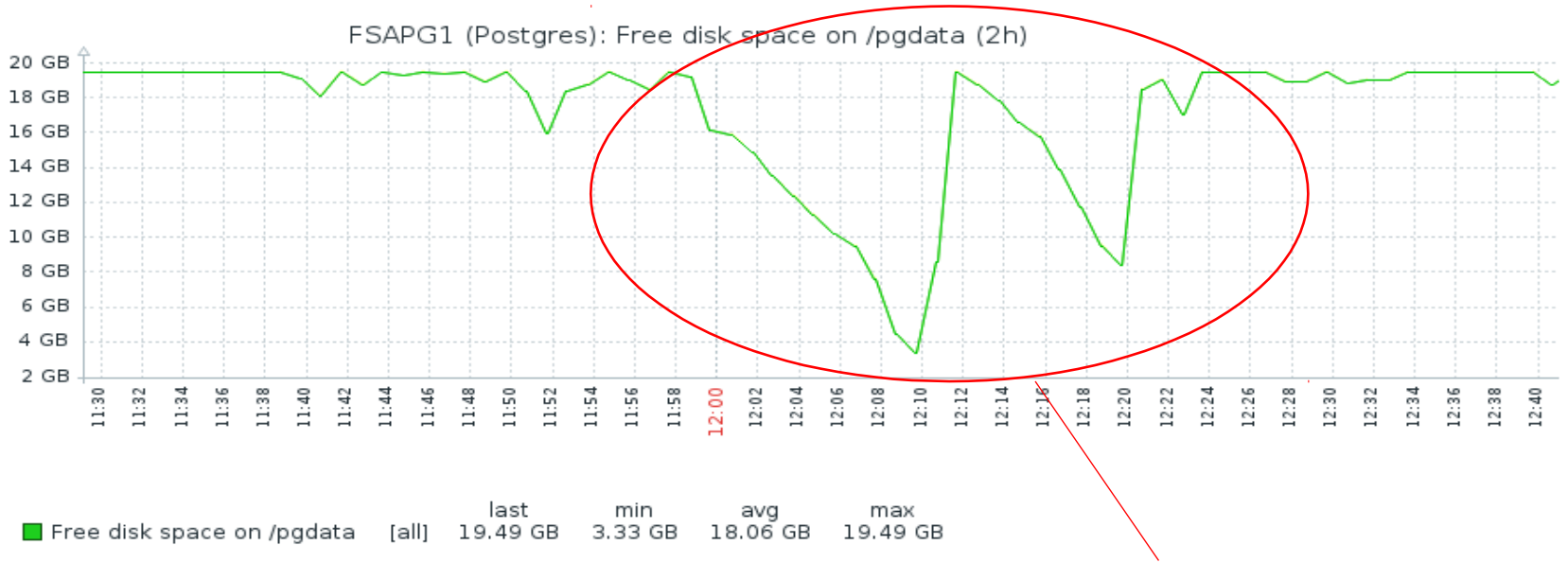
Для чего это нужно?

- Поиск запросов, вносящих наибольший вклад в нагрузку СУБД
- Информация о характере и масштабах активности на объектах БД
- Сравнение профилей нагрузки в разное время
- Помощь в расследовании инцидентов
- Анализ результатов нагрузочного тестирования
- …

Что для этого нужно?

- Хранение исторических данных
- Отчет с объективными значениями метрик за *произвольный* период времени
- *Минимальное* воздействие на наблюдаемую СУБД
- Простая установка и использование инструмента
- Достаточно собирать статистику по наиболее нагруженным запросам и объектам

Классический пример



Что это было?

Существующие средства

- PoWA (pg_stat_statements+ на python с Web-UI)

  Состоит из основного модуля и отдельного модуля для отображения UI

- Pgcluu

  Строит отличные отчеты по фиксированным интервалам, зато с системными метриками

- pgBager

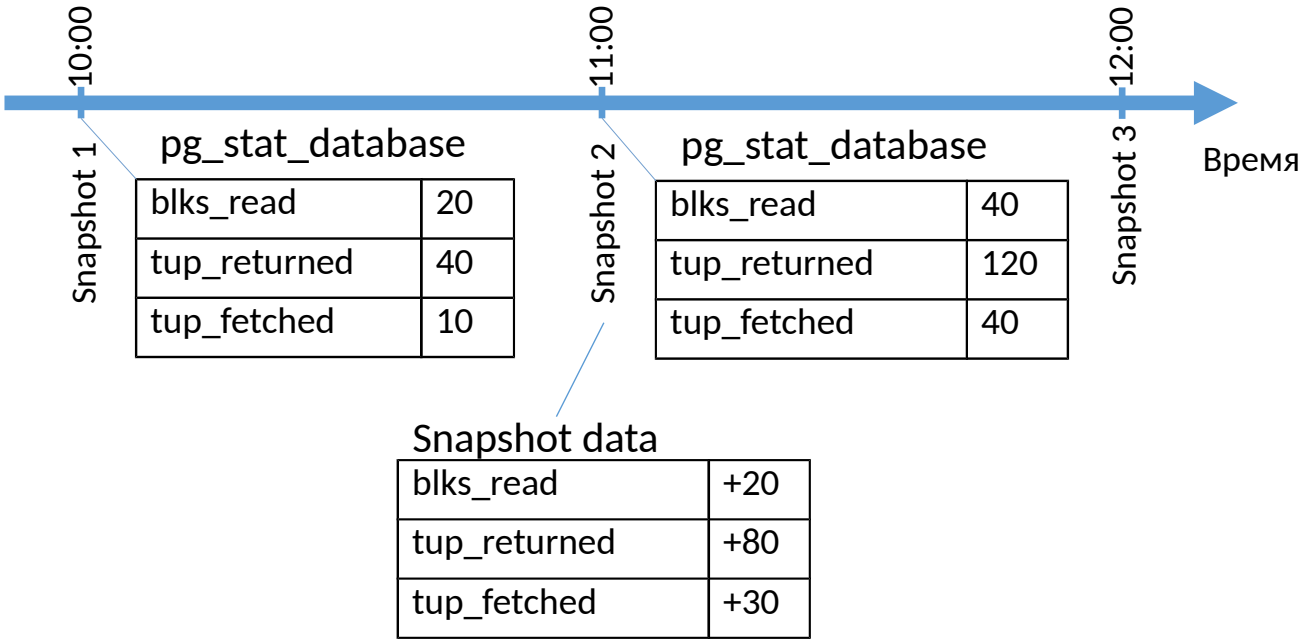
  Представляет собой анализатор журналов сервера, т.е. требует детального журналирования.

Идея *pg_profile*

pg_stat_database

Column	Type
datid	oid
datname	name
numbackends	integer
xact_commit	bigint
xact_rollback	bigint
blks_read	bigint
blks_hit	bigint
tup_returned	bigint
tup_fetched	bigint
tup_inserted	bigint
tup_updated	bigint
tup_deleted	bigint
conflicts	bigint
temp_files	bigint
temp_bytes	bigint
deadlocks	bigint
blk_read_time	double precision
blk_write_time	double precision
stats_reset	timestamp with time zone

Собираем изменения значений счетчиков



Архитектура pg_profile

pg_profile – расширение, написанное на pl/pgsql, состоящее из:

- Репозитория для хранения снимков
- Механизма сбора данных для снимков
- Механизма построения отчетов

Источники данных

Представление	Краткое содержание
pg_stat_statements	Статистики по выполнениям выражений
pg_stat_user_tables, pg_stat_user_indexes	Статистики по таблицам и индексам: количество записей, вставок, удалений, последовательных и индексных сканирований, и др.
pg_statio_user_tables, pg_statio_user_indexes	Статистики ввода-вывода по таблицам и индексам
pg_stat_user_functions	Статистики по вызовам и времени выполнения функций
pg_stat_database	Статистики по базам данных: транзакции (подтверждения и откаты), чтения блоков (с диска и из буферов), временные файлы и др.
pg_stat_bgwriter	Статистика процессов: checkpointer, background writer; статистика записи, выполненной backend’ами

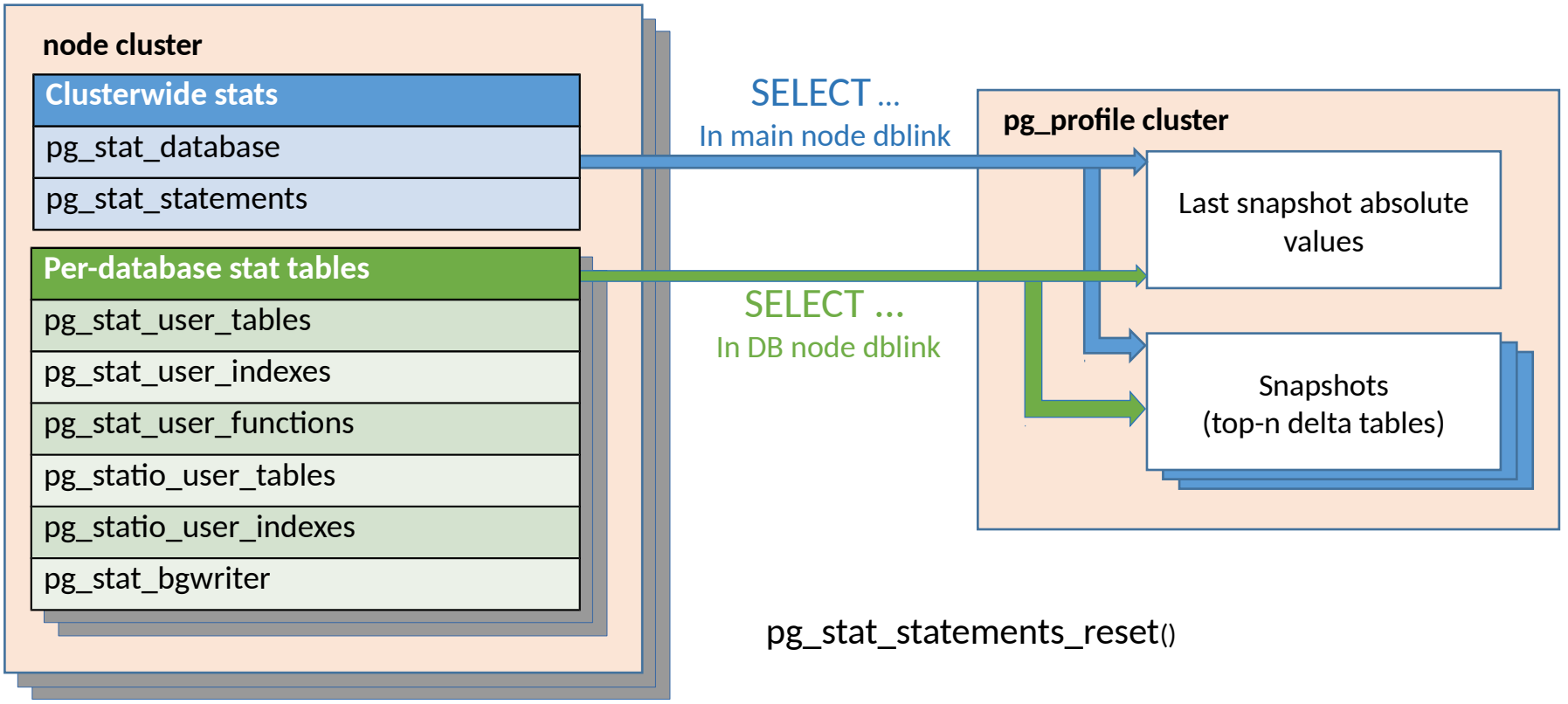
Когда кластеров много

pg_profile собирает статистики с нескольких кластеров:

- *node_new(node, connstr, node_enabled = true)*
- *node_drop(node)*
- *node_enable(node)*
- *node_disable(node)*
- *node_connstr(node, new_connstr)*
- *node_show()*

Enabled-ноды включены в *snapshot*(), остальные - *snapshot*(*node_name*)

SELECT snapshot();



cron:

```
*/30 * * * *   psql -d postgres -c 'SELECT profile.snapshot()'
```

Baselines

Baselines служат для изменения политики сохранения определенных снимков

- baseline_new([*node*,] *baseline_name*, *start_id*, *end_id*[, *days*])
- baseline_drop([*node*,] *name*)
- baseline_keep([*node*,] *name*, *days*)
- baseline_show([*node*])

Также применяются для построения отчетов

Построение отчетов

Получение списка снимков:

snapshot_show([*node_name*,] *days*)

```
postgres=# select * from profile.snapshot_show(1); -- за 1 сутки (local)
 snapshot |       date_time
----------+------------------------
...
      480 | 2017-09-11 10:00:01+03
      481 | 2017-09-11 10:30:01+03
      482 | 2017-09-11 11:00:01+03
```

Или просто выполнить запрос к таблице *snapshots*

Построение отчетов

Построение отчета:

```
$ psql -qtc "SELECT profile.report(480,482)" -o report_480_482.html
```

Функции построения отчетов:

- report([*node*,] *start_id*, *end_id*)
- report([*node*,] *baseline_name*)

Функции возвращают HTML-отчет, пригодный для просмотра в браузере

Разделы отчета

Отчет на примере pgbench

```
$ pgbench -T 40 -c 10 pgbench
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 40 s
number of transactions actually processed: 46845
latency average = 8.540 ms
tps = 1170.892287 (including connections establishing)
tps = 1170.947862 (excluding connections establishing)
```

Сравнительные отчеты

Сравнение нагрузки двух временных интервалов в одном отчете:

- report_diff([*node*,] *start1_id*, *end1_id*, *start2_id*, *end2_id*)
- report_diff([*node*,] *baseline1_name*, *baseline2_name*)
- report_diff([*node*,] *baseline1_name*, *start2_id*, *end2_id*)
- report_diff([*node*,] *start1_id*, *end1_id*, *baseline2_name*)

Отчет имеет те же разделы, но с информацией из двух интервалов

Сравнительные отчеты

Построим сравнительный отчет на примере второго *pgbench*

```
$ pgbench -T 80 -c 10 pgbench
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1
query mode: simple
number of clients: 10
number of threads: 1
duration: 80 s
number of transactions actually processed: 126006
latency average = 6.350 ms
tps = 1574.910909 (including connections establishing)
tps = 1574.948731 (excluding connections establishing)
```

Особенности установки

- Самый простой вариант

```
postgres=# CREATE EXTENSION pg_profile CASCADE;
```

- Аккуратный вариант

```
postgres=# CREATE EXTENSION dblink;
postgres=# CREATE EXTENSION pg_stat_statements;
postgres=# CREATE SCHEMA profile;
postgres=# CREATE EXTENSION pg_profile SCHEMA profile;
```

Параметры, вляющие на собираемые статистики

pg_stat_statements:

- `pg_stat_statements.max = 1000` (сколько надо)
- `pg_stat_statements.track = top/all`
- `pg_stat_statements.save = off`

Statistics Collector:

- `track_counts = on`
- `track_io_timing = on` (осторожно, *pg_test_timing*)
- `track_functions = pl/all`

Параметры

- Количество объектов в отсортированных таблицах (и снимках):

  pg_profile.topn = 20

- Время хранения снимков (в сутках)*:

  pg_profile.retention = 7

  * Используйте *baselines* для длительного хранения

Не без проблем…

- Сбор статистик в конце выполнения выражения
- *pg_stat_statements.track = all* собирает запросы всех уровней в одно представление, создавая проблемы с полем *%Total*
- *pg_relation_size()* может помешать сбору статистик снимка
- Хранение в снимках только *topn* записей может быть причиной неточности результатов
- Переполнение *pg_stat_statements*

Спасибо за внимание!

https://github.com/zubkov-andrei/pg_profile

PGConf.Russia 2020

Андрей Зубков

Администратор баз данных

Email: zubkov@moonset.ru