Postperf: performance en PostgreSQL

Abel Camarillo <acamari@verlet.org>

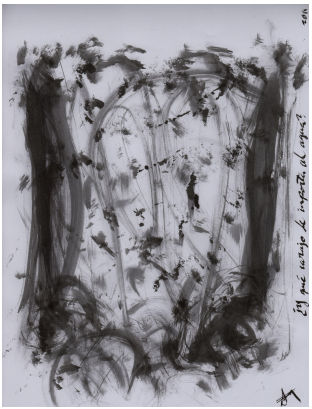
12 de abril de 2019

Agenda

- ¿Quién soy?
- ¿Qué es PostgreSQL?
- Cómo medir tiempos actuales
- Modelo de capas propuesto

¿Quién soy?

- Desarrollador de software desde el 2008 - OpenBSD, perl, C, sh, js
- Lead developer en Neuroservices Communications durante 6 años.
- Desarrollador freelance desde el 2015 - Verlet.
- Maintainer de 21 paquetes en el árbol oficial de OpenBSD - http://openports.se/bbmaint.php?\maint=acamari@verlet.org
- Interés en UNIX, poesía, ~arte~, cocina, etc...



¿Y qué carajo le importa al agua? - 2011

¿Qué es PostgreSQL?

- También llamado sólo postgres.
- De UCB - berkley, basado en Ingres (1985), licencia BSD-like
- Features avanzados: gist, geqo, sp-gist, gin
- Altamente personalizable:
  - C, SQL, PL/pgSQL, PL/Perl, PL/Lua, PL/Python, PL/V8, PL/sh, etc...
  - FDW - foreign data wrappers (conexiones a otras DB)
  - Publish/suscribe async - LISTEN/NOTIFY;
- JSON nativo: store, búsquedas indexadas(!), ¿web scale~?
- intersección de geometrías, rangos - indexada

Cómo medir tiempos actuales

psql \timing

```
$ psql –U postgres  db
psql (9.4.1)
Type ”help” for help.

db=# \timing on
Timing is on.
db=# select 1;
 ?column?
----------
        1
(1 row)

Time: 1.641 ms
db=#
```

pros: rápido, disponible para TODOS los queries.

cons: el tiempo incluye latencia de red y procesamiento del cliente.

Cómo medir tiempos actuales

EXPLAIN ANALYZE

```
db=# explain analyze select 1;
                                        QUERY PLAN
-----------------------------------------------------------------------------------
 Result  (cost=0.00..0.01 rows=1 width=0) (actual time=0.003..0.004 \
                                                  rows=1 loops=1)
 Planning time: 0.031 ms
 Execution time: 0.036 ms
(3 rows)
db=#
```

pros: excluye i/o cliente, incluye planeación y ejecución, granularidad

cons: no disponible en todos los queries (pero sí select/ins/upd/del)

Cómo medir tiempos actuales

Cualquier mecanismo que tenga su lenguaje/aplicación

Modelo de capas propuesto

Se propone el siguiente orden para analizar problemas de rendimiento, no necesariamente se tienen que cumplir todas las capas...

- Capa 0: física/hardware/fierros: beneficios N
- Capa 1: postgres.conf: beneficios N
- Capas SQL:
  - Capa 2: EXPLAIN/EXPLAIN ANALYZE
  - Capa 3: índices: beneficios $O(\log N), O(n \log N), O(< N)$
  - Capa 4: query: beneficios $O(\log N), O(n \log N), O(\ll N)$

Modelo de capas propuesto: capa 0 física

Al trabajar en esta capa:

pros: no hay que modificar la aplicación

cons: no esperar beneficios mayores que aritméticos, en el mejor de los casos

CPU: más ghz, más cores, menos threads, más cpus, más cache.

pros: beneficios lineales, cpu doble de rápido, db doble de rápida.

cons: beneficios lineales, costo no lineal$\sim$

Modelo de capas propuesto: capa 0 física

RAM: más ram, más rápida, doble/quádruple de canales - dual chan vs quad chan (Intel Core i7-5960X, AMD Opteron 6300-series 'Abu Dhabi' (32 nm), etc)

pros: beneficios lineales

cons: beneficios lineales, costo no lineal$\sim$, pocos órdenes de magnitud (x2, x4, x8, ... x16, x32)

Modelo de capas propuesto: capa 0 física

HDD (el cuello de botella común): más discos en arreglos RAID

pros: beneficios lineales, más órdenes de magnitud (x2, x4, ..., x128, x256), redundancia, datasets más grandes (cientos de TB)

cons: costo lineal + mantenimiento RAID, bottleneck se mueve a bus de datos, PCIev2 (4GBytes/s), PCIev3 (8GBytes/s)

HW RAID: pros: mucho más rápido, y robusto, libera CPU, topologías versátiles (discos duros por fibra a km del data center), independiente de SO - movible entre SOs, soporte de boot, transparente al SO

cons: más caro - tarjetas caras (>1000USD), cableado caro y de poco acceso, si la tarjeta falla, debe reemplazarse por otro modelo idéntico - o no se garantiza nada, incompatibilidad entre proveedores, pobre documentación.

Modelo de capas propuesto: capa 0 física

SW RAID: pros: barato, soporte de disciplinas de cifrado, si fallan todos los componentes menos los discos se pueden mover a otra computadora

cons: sin soporte boot (¿Linux?, OpenBSD sí soporta), incompatible entre SO, más carga al SO (CPU+RAM)

Modelo de capas propuesto: capa 1 postgresql.conf

Archivo principal de configuración de postgres en un cluster (instancia de postgres).

```
$ cd /tmp/
$ mkdir pg
$ initdb -D pg -E UTF-8 --no-locale -U postgres
[..]
$ ls -alsh pg/postgresql.conf
44 -rw-------  1 acamari  wheel  20.8K Oct 15 07:26 pg/postgresql.conf
$
```

Al trabajar en esta capa:

pros: No hay que modificar la aplicación, hay buena documentación, simple

cons: beneficios aritméticos, más complejo que meter fierros, más barato

Modelo de capas propuesto: capa 1 postgresql.conf

A modificar:

mem:

- `shared_buffers(128MB)`: global
- `temp_buffers(8MB)`: por sesión (CREATE TEMP TABLE)
- `work_mem(4MB)`: por sesión (`SELECT ... ORDER BY ...`, `SELECT ... WHERE col_con_indice_hash = const`)

pros: i/o a disco menos frecuente

cons: más uso de ram por sesión en algunos casos aunque no hagan nada, menos usuarios a la vez

Modelo de capas propuesto: capa 1 postgresql.conf

costos de planner: éstos se usan para decidir cuándo hacer seq scan vs index scan

- `seq_page_cost(1.0)`: aumentando: promueve idx scan, reduciendo: promueve seq scan
- `random_page_cost(4.0)`: lo opuesto$\sim$

Promover index scan:

pros: beneficia tiempos de ejecución dramáticamente en queries de alta 'selectividad' - que regresan pocos registros - al usar índices, complejidad $O(\log N), O(n \log N)$

cons: genera mucha más i/o para queries de baja selectividad - que regresan muchos registros

Modelo de capas propuesto: capa 1 postgresql.conf

vacumm: proceso periódico que regresa espacio en desuso a disco. MVCC

- `vacumm_cost_limit(200)`, cuando el costo de i/o en vacumming sea mayor a éste, duerme por `vacumm_cost_delay` milisegundos.
- `vacumm_cost_delay(0 [ms])`, si es 0 no se duerme nunca.

pros: vacumm durmiente genera i/o a disco más distribuido - en lugar de picos.

cons: si hay flujo agresivo de update/delete se acumulan 'zombies', no se tienen estadísticas tan frecuentes - que puede perjudicar planes, promover seq scan.

Modelo de capas propuesto: capa 1 postgresql.conf

wal: write ahead log: log de reconstrucción en caso de crash y para replicación.

- `checkpoint_segments(3)`, número de segmentos (de 16MB cada uno) a cachear antes de escribir en disco y al log WAL

Aumentando:

pros: i/o a disco menos frecuente, mejora rendimiento en picos

cons: más tiempo de recuperación de crash (pkill -9 postgres, kernel panic, pérdida de energía eléctrica), probable pérdida de datos.

Modelo de capas propuesto: capa 2 EXPLAIN ...

```
db=#
db=# create table herp (a int);
CREATE TABLE
db=# insert into herp (a) values (0), (1), (2), (3), (4), (5), (6), (7);
INSERT 0 8
db=#
db=# explain select a from herp where a = 3;
                      QUERY PLAN
------------------------------------------------------
 Seq Scan on herp  (cost=0.00..40.00 rows=12 width=4)
   Filter: (a = 3)
(2 rows)

db=# \timing
Timing is on.
db=#
db=#
db=#
```

```
db=# select a from herp where a = 3;
 a
___
 3
(1 row)

Time: 0.781 ms
db=#
db=# explain analyze select a from herp where a = 3;
                                                QUERY PLAN
_____________________________________________________________________________________________
 Seq Scan on herp  (cost=0.00..40.00 rows=12 width=4) (actual \
        time=0.020..0.024 rows=1 loops=1)
   Filter: (a = 3)
   Rows Removed by Filter: 7
 Planning time: 0.104 ms
 Execution time: 0.176 ms
(5 rows)

Time: 0.989 ms
```

```
db=#
db=# truncate herp;
TRUNCATE TABLE
Time: 36.185 ms
db=#
db=# -- insertar carga más real, 1 millón de reg
db=# insert into herp (a) select a from generate_series(1,1000 * 1000) as

INSERT 0 1000000
Time: 1399.472 ms
db=#
db=# explain select a from herp where a = 3;
                       QUERY PLAN
-------------------------------------------------------------
 Seq Scan on herp  (cost=0.00..17700.00 rows=1 width=4)
   Filter: (a = 3)
(2 rows)

Time: 0.811 ms
db=#
```

```
db=# explain analyze select a from herp where a = 3;
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on herp  (cost=0.00..17700.00 rows=1 width=4) (actual \
        time=0.036..205.402 rows=1 loops=1)
   Filter: (a = 3)
   Rows Removed by Filter: 999999
 Planning time: 0.104 ms
 Execution time: 205.452 ms
(5 rows)

Time: 206.054 ms
db=#
db=# -- limpiar tabla e insertar 10 millones
INSERT 0 10000000
Time: 13153.689 ms
db=#
db=# explain select a from herp where a = 3;
                         QUERY PLAN
-------------------------------------------------------------
```

```
 Seq Scan on herp  (cost=0.00..176992.00 rows=1 width=4)
   Filter: (a = 3)
(2 rows)

Time: 0.386 ms
db=#
db=# explain analyze select a from herp where a = 3;
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------------------------
 Seq Scan on herp  (cost=0.00..176992.00 rows=1 width=4) (actual \
        time=1.811..2379.152 rows=1 loops=1)
   Filter: (a = 3)
   Rows Removed by Filter: 9999999
 Planning time: 0.192 ms
 Execution time: 2379.197 ms
(5 rows)

Time: 2379.840 ms
db=#
db=#
```

```
db=# -- limpiar tabla e insertar 100 millones
INSERT 0 100000000
Time: 203101.817 ms
db=#
db=#
db=# explain select a from herp where a = 3;
                         QUERY PLAN
------------------------------------------------------------
 Seq Scan on herp  (cost=0.00..1769912.00 rows=1 width=4)
   Filter: (a = 3)
(2 rows)

Time: 1.197 ms
db=#
db=#
db=#
db=#
db=#
db=#
db=#
```

```
db=# -- disco a 180MBps un core al 30%
db=# explain analyze select a from herp where a = 3;
                                                     QUERY PLAN
---------------------------------------------------------------------------------------------------------------------
 Seq Scan on herp  (cost=0.00..1769912.00 rows=1 width=4) (actual \
        time=0.146..63725.876 rows=1 loops=1)
   Filter: (a = 3)
   Rows Removed by Filter: 99999999
 Planning time: 0.081 ms
 Execution time: 63725.905 ms
(5 rows)

Time: 63726.374 ms
db=#
```

Modelo de capas propuesto: capa 3 índices

Al trabajar en esta capa:

pros: no hay que modificar la aplicación, beneficios dramáticos, complejidad baja a

$$O(\log n) \text{ o } O(n * c \log n) | c \geq 1$$

dependiendo del tipo de índice btree, gin, gist, sp-gist, hash ($O(1)$ a $O(n)$)

cons: el indexado hace apreciablemente más lento (x2$\sim$) los insert/update/deletes debido al i/o extra, cada índice a una columna X gasta alrededor del almacenamiento en disco que ocupa la columna X en sí

```
db=# -- añadiendo índice btree a la columna principal
db=# create index on herp (a);
CREATE INDEX
Time: 148020.907 ms
db=#
db=# explain select a from herp where a = 3;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Index Only Scan using herp_a_idx on herp  (cost=0.57..8.59 \
       rows=1 width=4)
   Index Cond: (a = 3)
(2 rows)

Time: 3.252 ms
db=#
db=#
db=#
db=#
db=#
db=#
```

```
db=# -- leyendo de disco
db=# explain analyze select a from herp where a = 3;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Index Only Scan using herp_a_idx on herp  (cost=0.57..8.59 \
        rows=1 width=4) (actual time=32.697..32.704 rows=1 loops=1)
   Index Cond: (a = 3)
   Heap Fetches: 1
 Planning time: 0.208 ms
 Execution time: 33.379 ms
(5 rows)

Time: 34.234 ms
db=#
db=#
db=#
db=#
db=#
db=#
db=#
```

```
db=# -- dato aledaño, probablemente en cache
db=# explain analyze select a from herp where a = 50;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Index Only Scan using herp_a_idx on herp  (cost=0.57..8.59 rows=1 \
        width=4) (actual time=0.040..0.043 rows=1 loops=1)
   Index Cond: (a = 50)
   Heap Fetches: 1
 Planning time: 0.165 ms
 Execution time: 0.097 ms
(5 rows)

Time: 1.064 ms
db=#
db=#
db=#
db=#
db=#
db=#
db=#
```

```
db=# -- otra búsqueda
db=# explain analyze select a from herp where a = 50000000;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Index Only Scan using herp_a_idx on herp  (cost=0.57..8.59 rows=1 \
        width=4) (actual time=1.045..1.050 rows=1 loops=1)
   Index Cond: (a = 50000000)
   Heap Fetches: 1
 Planning time: 0.191 ms
 Execution time: 1.109 ms
(5 rows)

Time: 2.020 ms
db=#
```

Modelo de capas propuesto: capa 4 query

En esta capa nos podemos enfocar en optimizar queries específicos, también podemos migrar más trabajo de búsqueda de datos directo a la db - y reducir los ciclos aplicación - net i/o - pg (de ida y vuelta) y explotar la optimización de queries que hace postgres:

Todo en esta capa interfiere con la aplicación, hay que modificarla

Al trabajar en esta capa:

- Migrando a db la lógica: stored procedures/funciones, SQL, PL/pgSQL;

  pros: eliminación de mucha net/io, explota el optimizador de queries de postgres, permite reciclar lógica entre aplicaciones

  cons: modificar aplicación, requiere aprender sintaxis nueva

- Largas cadenas de selects/insert/update convertidas a un solo query mejora integridad de datos y no genera locking explícito

Modelo de capas propuesto: capa 4 query

Errores comunes: count(*)

Genera un seq scan de toda la tabla, si sólo se quiere saber si hay registros o no en una tabla, se sugiere:

```
db=# explain select count(*) from herp;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Aggregate  (cost=1692478.00..1692478.01 rows=1 width=0)
   ->  Seq Scan on herp  (cost=0.00..1442478.00 rows=100000000 width=0)
(2 rows)

Time: 0.854 ms
db=# select count(*) from herp;
   count
-----------
 100000000
(1 row)
Time: 27080.496 ms
```

```
db=#
db=# explain analyze select 1 from herp limit 1;
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Limit  (cost=0.00..0.01 rows=1 width=0) (actual time=0.035..0.037 \
        rows=1 loops=1)
   ->  Seq Scan on herp  (cost=0.00..1442478.00 rows=100000000 \
        width=0) (actual time=0.031..0.031 rows=1 loops=1)
 Planning time: 0.161 ms
 Execution time: 0.113 ms
(4 rows)

Time: 0.966 ms
db=#
db=# select 1 from herp limit 1;
 ?column?
----------
        1
(1 row)
```

```
Time: 0.801 ms
db=#
```

Modelo de capas propuesto: capa 4 query

Pérdida de integridad de datos, sin locking explícito, las transacciones no son tan atómicas ni aisladas ni consistentes.

De Wikipedia:

> In computer science, ACID (Atomicity, Consistency, Isolation, Durability) is a set of properties that guarantee that database transactions are processed reliably. In the context of databases, a single logical operation on the data is called a transaction. For example, a transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction.

```
db=# create table t1 (a int primary key, value int);
CREATE TABLE
db=# create table t2 (a int references t1, valuesum int);
CREATE TABLE
db=#
db=# insert into t1 values (0, 50), (0, 60), (1, 70), (1, 80), (1, 90);
INSERT 0 5
db=#
db=# begin;
BEGIN
db=# -- select optimista
db=# select  a, sum(value) from t1 group by a;
 a | sum
---+-----
 1 | 240
 0 | 110
(2 rows)

db=#
db=#
```

```
db=# -- concurrentemente:
db=# -- delete from t1 where (a = 1 and value = 90) or \
                                (a = 0 and value = 60);
db=# -- DELETE 2
db=# insert into t2 (a, valuesum) values (1, 240), (0, 110);
INSERT 0 2
db=#
db=# select * from t1; select * from t2;
 a | sum
---+-----
 1 | 150
 0 |  50
(2 rows)

 a | valuesum
---+----------
 1 |      240
 0 |      110
(2 rows)
db=# -- XXX NO CONCUERDAN!
```

```
db=# commit; -- sigh...
COMMIT
db=#
db=#
db=#
db=#
db=# truncate t2, t1;
TRUNCATE TABLE
db=# insert into t1 values (0, 50), (0, 60), (1, 70), (1, 80), (1, 90);
INSERT 0 5
db=#
db=# -- insert realmente atómico, no necesita ni BEGIN ni COMMIT
db=# -- concurrentemente:
db=# -- delete from t1 where (a = 1 and value = 90) or \
                                 (a = 0 and value = 60);
db=# -- DELETE 2
db=# insert into t2 select  a, sum(value) from t1 group by a;
INSERT 0 2
Time: 19.937 ms
db=#
```

```
db=# select * from t1; select * from t2;
 a | value
---+-------
 0 |    50
 1 |    70
 1 |    80
(3 rows)

 a | valuesum
---+----------
 1 |      150
 0 |       50
(2 rows)
db=#  -- INTEGRIDAD!: concuerdan
```

Modelo de capas propuesto: capa 4 query

- PL/pgSQL: pendiente.
- Tipos de datos personalizados: pendiente.

¿Preguntas?