Índices bloom

Posibles usos, resultados laboratorio

Lic. Fernando Fontana – nov/2017

Temario

- Me presento
- ¿Qué son los índices bloom?
- ¿Cómo usarlos?
- Posibles aplicaciones
- Tipos de datos y cómo extenderlos
- Resultados pruebas
- Conclusiones

Me presento

- DBA PostgreSQL y Oracle, consultor en integración software
- PostgreSQL press contact para Uruguay
- Clientes en Montevideo y Buenos Aires
- +15 años DBA Postgres para Statum para su producto Apia

- => espónsor y motivador de este trabajo

¿Qué son los índices bloom?

Postgres es muy rico en variedad de índices:

- Tipo btree (por defecto), hash, brin, gist/gin
- Parciales, por expresiones

Bloom es un nuevo tipo de índice

Disponible a partir versión 9.6 como módulo contrib (extensión)

Basados en filtros bloom: estructura datos probabilística para calcular si un elemento pertenece a conjunto. Devuelve falsos positivos, nunca falsos negativos. Usa funciones de hash. Inventado por Burton Bloom.

¿Qué son los índices bloom?

- Bloom: **Se usa para condiciones por igualdad sobre las columnas y en algunas o todas de las columnas indexadas.**

- Btree:

  - Ídem condiciones por igualdad, pero sólo sobre columnas “prefijo” => importa orden columnas en creación índice
  - Última condición del prefijo puede ser por <, <=, >, >=

¿Qué son los índices bloom?

```
create index indbloom on articulos using bloom
    (peso, altura, ancho, tipo, volumen, unidadesxcaja, precio);
create index indbtree on articulos
    (peso, altura, ancho, tipo, volumen, unidadesxcaja, precio);


select * from articulos where altura=10 and tipo = 'C';
=> puede usar bloom
select * from articulos where volumen = 10 and ancho = 15 and peso = 200;
=> puede usar bloom o btree por columna peso
select * from articulos where unidadesxcaja = 144;
=> puede usar bloom
select * from articulos where peso >= 100;
=> puede usar btree por columna peso
```

¿Qué son los índices bloom?

- Se logra funcionalidad similar creando varios btree
- Índice hash resuelve similar si se indican condiciones por igualdad en todas las columnas indexadas.
- Índice bloom es más chico que un btree.
- Obtiene filas candidatas que deben ser verificadas en la tabla (chequear si hay falsos positivos)

¿Qué son los índices bloom?

```
explain analyze select * from articulos where unidadesxcaja = 144

 Bitmap Heap Scan on articulos  (cost=202800.00..202808.00 rows=2 width=40) (actual
time=121.280..122.562 rows=2 loops=1) ⇐ 2 filas resultantes

   Recheck Cond: (unidadesxcaja = 144)

   Rows Removed by Index Recheck: 153  ⇐ detecta falsos positivos

   Heap Blocks: exact=155

   ->  Bitmap Index Scan on iart1  (cost=0.00..202800.00 rows=2 width=0) (actual
time=118.747..118.747 rows=155 loops=1) ⇐ construcción bitmap, calcula filas candidatas

         Index Cond: (unidadesxcaja = 144)

 Planning time: 0.152 ms

 Execution time: 122.585 ms
```

¿Cómo usarlos?

- Instalar contrib
- Crear extensión bloom
- ```
  create index nombreind on nombretabla
    using bloom (nombrecol1, nombrecol2, … )
    with (length=largo, col1=n1, col2=n2, …); ⇐ opcional
  ```
- largo es cantidad bits entrada índices
- *n1,n2, …* es cantidad bits lugares cálculo hash para columna 1,2, …
- Valores por defecto: largo=80, 2 bits por cada columna
- Ejemplo
  ```

Posibles aplicaciones

- Condiciones establecidas por un usuario en un formulario de búsqueda. No se sabe a priori cuáles casillas serán completadas.
- Atributos variables según tipo fila. Ejemplos:
  - Artículos: si es en caja: largo, ancho, peso; si es fruta/verdura: fecha envasado, precio por kilo; ...
  - Una persona, además de nro documento y nombre, si es empleado tiene hora entrada, hora salida y sueldo, si es contratado cantidad horas y valor hora, si es becario horas totales asignadas.
- Tabla común con varios índices: analizar posibilidad de fusionarlos en un único bloom.

Tipos de datos y cómo extenderlos

- Inicialmente índice sólo acepta columnas **int** y **text** ya que su uso más común es para comparar con constantes dadas por usuario.
- Para usarlo en otros tipos de datos, crear “operator class” con una función de hash.
- Ejemplo: para numeric:

```
create operator class numeric_ops
       default for type numeric using bloom as
          operator 1 =(numeric,numeric),
          function 1 hash_numeric(numeric);
```

Tipos de datos y cómo extenderlos

- Para date no hay función de hash, crear una por ejemplo como:

```
create function hashdate(d date) returns integer as $$
begin return hashtext(d::text); end;
$$ language plpgsql immutable;

create operator class date_ops
default for type date using bloom as
    operator 1 =(date,date),
    function 1 hashdate(date);
```

- Ver funciones hash existentes desde psql con **`\df+ hash*`**

Resultados pruebas

- Se efectuaron pruebas de laboratorio para:
  - Ver performance
  - Tratar de encontrar valores óptimos para la configuración de un índice (largo y #bits cada columna)
- Tabla generada con 10 a 100 millones de filas, valores aleatorios.
- Índice con 4 a 9 columnas indexadas, con largos entre 40 y 400 bits y #bits por columna entre 1 y 15 bits.
- Consultas por una o varias condiciones.

Resultados pruebas

- Configurar work_mem >= 32mb para evitar que bitmap sea “lossy” (guarda información de bloques y no de filas participantes)

```
Bitmap Heap Scan on tb  (cost=915549.22..933518.00 rows=4900 width=40) (actual time=76662.708..76662.708
  rows=0 loops=1)

  Recheck Cond: (c1 = 123456)

  Rows Removed by Index Recheck: 47571259

  Heap Blocks: exact=21801 lossy=394828

  ->  Bitmap Index Scan on indbloom  (cost=0.00..915548.00 rows=4900 width=0) (actual
  time=2512.228..2512.228 rows=3686802 loops=1)
```

Resultados pruebas

- Configuración índice:
  - Valores por defecto no son los mejores.
  - largo >= 120. Valor muy grande penaliza creación bitmap. Exagerando, prefiere un **`seq scan`** a la tabla. Redondea a múltiplo 16. Ver tamaño índice resultante.
  - 3 <= #bits columna <= 15. Valores más grandes no mejoran cantidad filas candidatas, y llegan a aumentarlo. Valores más chicos crean muchas filas candidatas. Probar en cada caso valores que se adapten mejor. Varía según selectividad columna, cantidad columnas indexadas.
  - Aumentando largo o #bits aumenta tiempo creación y mantenimiento índice.

Conclusiones

- Postgres tiene varios tipos de índices: analizar posibilidad de usar índices no btree (hash, bloom, brin, gist/gin, parciales, indexar expresiones).
- ¿Condiciones variables por igualdad? Candidato a usar bloom.
- Probar bloom en cada caso con distintos parámetros según selectividad columnas, cantidad de filas. Experimentar, probar con datos reales, ¡no quedarse con valores por defecto!
- Analizar reemplazar uno o varios btree por un índice bloom.
- Escasa documentación, por detalles ver fuente.

Conclusiones

- Configurar work_mem, al menos 32mb
- length >= 120, #bits, 3 <= columna <= 15 razonables según #valores distintos
- Comprobar uso mediante:
  - explain
  - pg_stat_user_indexes
- En discos ssd funcionan mejor que en hdd

Bibliografía

- Documentación postgres:
  https://www.postgresql.org/docs/9.6/static/bloom.html
- So what are Bloom indexes for Postgres? :
  http://www.cybertec.at/so-what-are-bloom-indexes-for-postgres/
- Código fuente:
  https://github.com/postgres/postgres/tree/master/contrib/bloom
- Choosing bloom index parameters:
  http://blog.coelho.net/database/2016/12/11/postgresql-bloom-index.html
- Filtros bloom: https://en.wikipedia.org/wiki/Bloom_filter

Gracias

¿Preguntas?

fernando.fontana@gmail.com