PostgreSQL数据库内核探究

MengQingzhong.com

(https://www.mengqingzhong.com/)

PostgreSQL索引(8) – RUM

2020-10-01 (Https://Www.Mengqingzhong.Com/2020/10/01/Postgresql-Index-Rum-8/) 孟庆钟 (Https://Www.Mengqingzhong.Com/Author/Mengqingzhong/)

本文翻译自Egor Rogov的英文博客 (https://habr.com/en/company/postgrespro/blog/452116/)，且已征得Egor Rogov的同意。译者做了少量修改。

审校：纪昀红，中国人民大学信息学院在读博士生。

在前面几篇文章中，我们讨论了通用索引引擎、AM的接口和五种AM（Hash索引、B-tree索引、GiST索引、SP-GiST索引、GIN索引）。这篇文章将介绍RUM索引。

下一代GIN索引被称为RUM。

它扩展了GIN索引中的概念，可以使我们更快地进行全文检索。在本系列介绍的所有AM中，它是唯一一个没有包含在PG标准安装包中的AM，它是一个插件。可以通过下面几种方式获得：

- 使用yum或apt从PGDG源安装 (https://www.postgresql.org/download/)。例如，假如从postgresql-10包中安装了PG10，就安装postgresql-10-rum。
- 使用源代码自行编译
- 使用Postgres Pro Enterprise

1. GIN的缺点

RUM超越了GIN索引的什么限制呢?

首先，tsvector类型不但包含词素，它还包含词素在文档中的位置信息。就像上次描述的那样，GIN索引不存储这些信息。由于这个原因，在PG 9.6中使用GIN索引搜索短语时，效率非常低，它需要访问原始的数据进行recheck。

其次，搜索系统通常按照某种相关性（无论它是什么意思）返回结果。为了达到这个目的，我们可以使用ts_rank和ts_rank_cd两个函数。但是它需要为每行结果进行计算，效率很低。

大致上来说，RUM可以被看成是添加了位置信息的GIN的索引，并且可以按照所需的顺序返回结果（就像GiST中的KNN查询）。让我们逐步分析。

2. 搜索短语

一个全文检索的查询可以包含特殊的操作符，用来考虑词素之间的距离，例如，查找hand和thigh被两个单词分割的文档。

```
select to_tsvector('Clap your hands, slap your thigh') @@
                  to_tsquery('hand <3> thigh');
```

```
 ?column?
----------
 t
(1 row)
```

或者，我们指示一个单词必须在另一个的后面。

```
select to_tsvector('Clap your hands, slap your thigh') @@
                  to_tsquery('hand <-> slap');
```

```
 ?column?
----------
 t
(1 row)
```

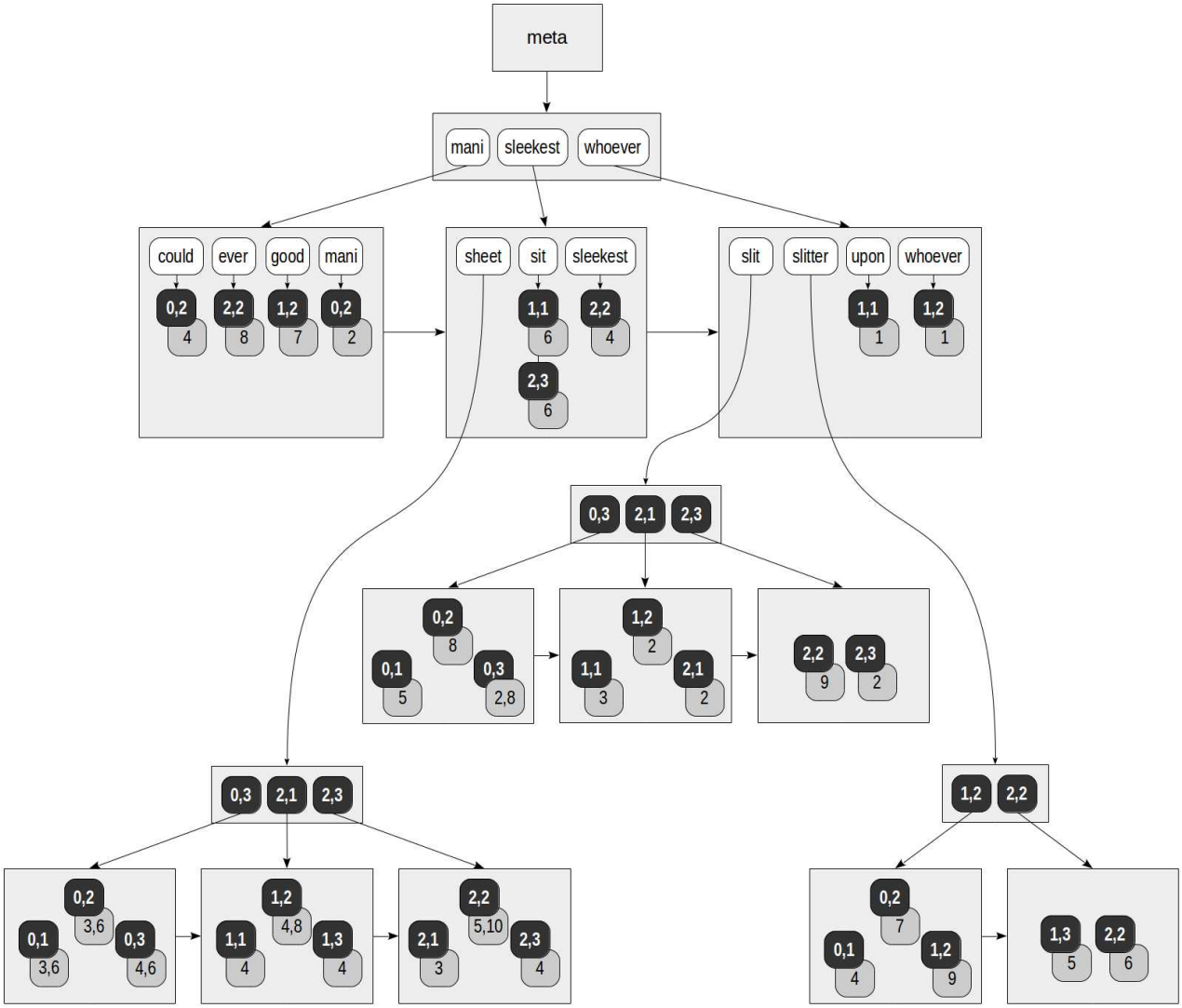
常规的GIN索引，可以返回包含两个词素的文档，但是我们只能通过查询tsvector来计算它们之间的距离。

```
select to_tsvector('Clap your hands, slap your thigh');
```

```
            to_tsvector
-----------------------------------
 'clap':1 'hand':3 'slap':4 'thigh':6
(1 row)
```

在RUM索引中，每个词素不仅指向基表的行：每个TID和一个链表绑定在一起，这个链表包含了每个词素在文档中出现的位置。上篇文章中的例子，使用RUM索引时，树的形状可能像下面这样。tsvector类型默认使用rum_tsvector_ops操作符类。

```
create extension rum;
create index on ts using rum(doc_tsv);
```



(https://www.mengqingzhong.com/wp-content/uploads/2020/12/wp_editor_md_2926bc9a8ceb170eed75fe0492259d18.jpg)

灰色方块表示位置信息。

```
select ctid, left(doc,20), doc_tsv from ts;
```

```
  ctid |         left         |                          doc_tsv
-------+----------------------+-------------------------------------------------------
---
 (0,1) | Can a sheet slitter  | 'sheet':3,6 'slit':5 'slitter':4
 (0,2) | How many sheets coul | 'could':4 'mani':2 'sheet':3,6 'slit':8 'slitter':7
 (0,3) | I slit a sheet, a sh | 'sheet':4,6 'slit':2,8
 (1,1) | Upon a slitted sheet | 'sheet':4 'sit':6 'slit':3 'upon':1
 (1,2) | Whoever slit the she | 'good':7 'sheet':4,8 'slit':2 'slitter':9 'whoever':1
 (1,3) | I am a sheet slitter | 'sheet':4 'slitter':5
 (2,1) | I slit sheets.       | 'sheet':3 'slit':2
 (2,2) | I am the sleekest sh | 'ever':8 'sheet':5,10 'sleekest':4 'slit':9 'slitte
r':6
 (2,3) | She slits the sheet  | 'sheet':4 'sit':6 'slit':2
(9 rows)
```

GIN提供了fastupdate参数，RUM删除了这个参数。

我们继续使用pgsql-hackers邮件列表作为示例。

```
alter table mail_messages add column tsv tsvector;
set default_text_search_config = default;
update mail_messages
set tsv = to_tsvector(body_plain);
```

```
...
UPDATE 356125
```

使用GIN索引时，查询是这样执行的：

```
create index tsv_gin on mail_messages using gin(tsv);
explain (costs off, analyze)
select * from mail_messages
where tsv @@ to_tsquery('hello <-> hackers');
```

```
                                QUERY PLAN
---------------------------------------------------------------------------
 Bitmap Heap Scan on mail_messages (actual time=2.490..18.088 rows=259 loops=1)
   Recheck Cond: (tsv @@ to_tsquery('hello <-> hackers'::text))
   Rows Removed by Index Recheck: 1517
   Heap Blocks: exact=1503
   ->  Bitmap Index Scan on tsv_gin (actual time=2.204..2.204 rows=1776 loops=1)
         Index Cond: (tsv @@ to_tsquery('hello <-> hackers'::text))
 Planning time: 0.266 ms
 Execution time: 18.151 ms
(8 rows)
```

从计划中可以看出，GIN索引被使用了，但是它返回了1776条候选，最终被淘汰1517条，只剩下259条。

让我们删除GIN索引，建立RUM索引。

```
drop index tsv_gin;
create index tsv_rum on mail_messages using rum(tsv);
```

这个索引包含了所有所需的信息，查询非常精确：

```
explain (costs off, analyze)
select * from mail_messages
where tsv @@ to_tsquery('hello <-> hackers');
```

```
                                   QUERY PLAN
-------------------------------------------------------------------------------
 Bitmap Heap Scan on mail_messages (actual time=2.798..3.015 rows=259 loops=1)
   Recheck Cond: (tsv @@ to_tsquery('hello <-> hackers'::text))
   Heap Blocks: exact=250
   ->  Bitmap Index Scan on tsv_rum (actual time=2.768..2.768 rows=259 loops=1)
         Index Cond: (tsv @@ to_tsquery('hello <-> hackers'::text))
 Planning time: 0.245 ms
 Execution time: 3.053 ms
(7 rows)
```

3. 按相关性排序

为了快速按照所需的顺序返回文档，RUM支持排序操作符，在GiST文档中描述过。RUM插件定义了这个操作符，<=>，它可以返回文档(tsvector)和查询(tsquery)之间的距离。例如：

```
select to_tsvector('Can a sheet slitter slit sheets?')
       <=> to_tsquery('slit');
```

```
 ?column?
----------
  16.4493
(1 row)
```

```
select to_tsvector('Can a sheet slitter slit sheets?')
       <=> to_tsquery('sheet');
```

```
 ?column?
----------
  13.1595
(1 row)
```

这个文档看起来与第二个文档比与第一个文档更加相关：一个单词出现的次数越多，这个文档价值越低。

我们再一个相对大一点的数据比较GIN和RUM：找出包含hello和hackers的十篇最相关的文档。

```
explain (costs off, analyze)
select * from mail_messages
where tsv @@ to_tsquery('hello & hackers')
order by ts_rank(tsv,to_tsquery('hello & hackers'))
limit 10;
```

```
                                                                  QUERY PLAN
----------------------------------------------------------------------------------------------
--------
 Limit (actual time=27.076..27.078 rows=10 loops=1)
   ->  Sort (actual time=27.075..27.076 rows=10 loops=1)
         Sort Key: (ts_rank(tsv, to_tsquery('hello & hackers'::text)))
         Sort Method: top-N heapsort  Memory: 29kB
         ->  Bitmap Heap Scan on mail_messages (actual ... rows=1776 loops=1)
               Recheck Cond: (tsv @@ to_tsquery('hello & hackers'::text))
               Heap Blocks: exact=1503
               ->  Bitmap Index Scan on tsv_gin (actual ... rows=1776 loops=1)
                     Index Cond: (tsv @@ to_tsquery('hello & hackers'::text))
 Planning time: 0.276 ms
 Execution time: 27.121 ms
(11 rows)
```

GIN索引返回了1776条，接着使用一个单独的步骤排序，选出十个最好的匹配。

借助RUM索引，查询可以通过一个简单的索引扫描完成：无需额外查看文档，无需额外的排序：

```
explain (costs off, analyze)
select * from mail_messages
where tsv @@ to_tsquery('hello & hackers')
order by tsv <=> to_tsquery('hello & hackers')
limit 10;
```

```
                                                            QUERY PLAN
----------------------------------------------------------------------------------------------
-------
 Limit (actual time=5.083..5.171 rows=10 loops=1)
   ->  Index Scan using tsv_rum on mail_messages (actual ... rows=10 loops=1)
         Index Cond: (tsv @@ to_tsquery('hello & hackers'::text))
         Order By: (tsv <=> to_tsquery('hello & hackers'::text))
 Planning time: 0.244 ms
 Execution time: 5.207 ms
(6 rows)
```

4. 附加字段

RUM索引，和GIN索引一样，可以建立在多个字段上。但是GIN索引把每个列的词素单独存储。RUM可以把一个附加属性附加到一个主属性上。这可以借助rum_tsvector_addon_ops操作符类完成。

```
create index on mail_messages
  using rum(tsv RUM_TSVECTOR_ADDON_OPS, sent)
  WITH (ATTACH='sent', TO='tsv');
```

我们可以使用这个索引对结果按照附加的属性进行排序：

```
select id, sent, sent <=> '2017-01-01 15:00:00'
from mail_messages
where tsv @@ to_tsquery('hello')
order by sent <=> '2017-01-01 15:00:00'
limit 10;
```

```
   id    |        sent         | ?column?
---------+---------------------+----------
 2298548 | 2017-01-01 15:03:22 |      202
 2298547 | 2017-01-01 14:53:13 |      407
 2298545 | 2017-01-01 13:28:12 |     5508
 2298554 | 2017-01-01 18:30:45 |    12645
 2298530 | 2016-12-31 20:28:48 |    66672
 2298587 | 2017-01-02 12:39:26 |    77966
 2298588 | 2017-01-02 12:43:22 |    78202
 2298597 | 2017-01-02 13:48:02 |    82082
 2298606 | 2017-01-02 15:50:50 |    89450
 2298628 | 2017-01-02 18:55:49 |   100549
(10 rows)
```

我们在这个查询中，查询了距离指定日期最近的行，无论早于还是晚于指定日期。如果想要查找严格早于或晚于指定日期时，应该使用<=|(o或|=>)操作符。正如所期待的那样，查询只是需要一个简单的索引扫描：

```
explain (costs off)
select id, sent, sent <=> '2017-01-01 15:00:00'
from mail_messages
where tsv @@ to_tsquery('hello')
order by sent <=> '2017-01-01 15:00:00'
limit 10;
```

```
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 Limit
   ->  Index Scan using mail_messages_tsv_sent_idx on mail_messages
         Index Cond: (tsv @@ to_tsquery('hello'::text))
         Order By: (sent <=> '2017-01-01 15:00:00'::timestamp without time zone)
(4 rows)
```

如果没有这个附加的属性，我们需要把索引扫描的结果使用Sort算子排序。

除了日期类型，我们当然可以附加其它数据类型的字段。几乎所有的基本类型都支持。

5. 其它操作符类

首先看rum_tsvector_hash_ops和rum_tsvector_hash_addon_ops。它们与已经介绍过的rum_tsvector_ops和rum_tsvector_addon_ops很相似，只是它们存储词素的Hash值，而不是词素本身。这可以减小索引的体积，当然查询变得不精确，并且需要重新检查。而且，这导致无法支持部分匹配。

rum_tsquery_ops非常有趣，它们帮助我们解决一个相反的问题：找出符合文档的查询。为什么有这个需求？假如一个用户订阅了某种新的商品或者每种类型的文档，当新出现这种文档时，应该推送给用户。例如：

```
create table categories(query tsquery, category text);
insert into categories values
  (to_tsquery('vacuum | autovacuum | freeze'), 'vacuum'),
  (to_tsquery('xmin | xmax | snapshot | isolation'), 'mvcc'),
  (to_tsquery('wal | (write & ahead & log) | durability'), 'wal');
create index on categories using rum(query);
select array_agg(category)
from categories
where to_tsvector(
  'Hello hackers, the attached patch greatly improves performance of tuple
   freezing and also reduces size of generated write-ahead logs.'
) @@ query;
```

```
  array_agg
--------------
 {vacuum,wal}
(1 row)
```

rum_anyarray_ops和rum_anyarray_addon_ops为了数组而设计，已经在GIN索引中介绍过，此处不再赘述。

6. 索引体积和WAL日志量

因为RUM存储的信息比GIN更多，它的体积也更大。上一篇文章已经比较过体积，现在把把RUM加进去：

```
  rum   |  gin   |  gist  | btree
--------+--------+--------+--------
 457 MB | 179 MB | 125 MB | 546 MB
```

可以看到，体积增大了很多，这是快速查找所需的代价。

值得注意的是：RUM是一个插件，因此安装它并不需要对内核进行任何修改。需要解决的一个问题是日志记录里的生成问题。PG的Xlog子系统必须要绝对地可靠，因此不允许插件自己写Xlog。写Xlog的过程需要交给PG内核。插件首先通知内核想要修改页面，然后修改页面，最后通知内核已经修改完毕。内核根据页面修改前后的编号生成Xlog。

现在的日志生成算法会按字节比较页面，检测出需要更新的段，记录每个段。当更新页面中少量字节时，或整个页面时这种方法还可以。但是需要对页面多处进行修改时，实际记录的Xlog量比实际修改的字节数要多很多。

由于这个原因，频繁更新RUM索引可能会产生比GIN（不是插件，而是内核的一部分，以它自己的方式管理日志）索引多的多的日志。这个现象与具体的负载有关。我们可以进行一些操作，记录一下日志产生的数量。在开始和结束之前，使用pg_current_wal_location（PG 9.x版本中为pg_current_xlog_location）函数获取当前Xlog的位置，然后计算日志量。

当然，我们需要考虑很多方面。我们需要保证只有一个用户在使用这个系统，否则会产生额外的日志记录。即使这样，我们也需要考虑，除了RUM，更新表本身也会产生日志。一些参数也会影响日志量，下面使用replica日志级别、没有压缩。

```
select pg_current_wal_location() as start_lsn \gset
insert into mail_messages(parent_id, sent, subject, author, body_plain, tsv)
  select parent_id, sent, subject, author, body_plain, tsv
  from mail_messages where id % 100 = 0;
```

```
INSERT 0 3576
```

```
delete from mail_messages where id % 100 = 99;
```

```
DELETE 3590
```

```
vacuum mail_messages;
```

```
insert into mail_messages(parent_id, sent, subject, author, body_plain, tsv)
  select parent_id, sent, subject, author, body_plain, tsv
  from mail_messages where id % 100 = 1;
```

```
INSERT 0 3605
```

```
delete from mail_messages where id % 100 = 98;
```

```
DELETE 3637
```

```
vacuum mail_messages;
insert into mail_messages(parent_id, sent, subject, author, body_plain, tsv)
  select parent_id, sent, subject, author, body_plain, tsv from mail_messages
  where id % 100 = 2;
```

```
INSERT 0 3625
```

```
delete from mail_messages where id % 100 = 97;
```

```
DELETE 3668
```

```
vacuum mail_messages;
select pg_current_wal_location() as end_lsn \gset
select pg_size_pretty(:'end_lsn'::pg_lsn - :'start_lsn'::pg_lsn);
```

```
 pg_size_pretty
----------------
 3114 MB
(1 row)
```

日志大约为3GB，如果使用GIN索引重复试验，只生成大约700MB。

因此，很有必要研发一种不同的算法，可以用来找出插入和删除最小的差异，可以把页面的状态转换成另外一个状态。diff使用类似的工作方式。Oleg Ivanov实现了一个这种算法，社区正在讨论，在上面的例子中，日志里减小到1900M，代价时性能稍慢一些。

不幸的是，这个patch停止开发了。

7. 属性

和往常一样，看一下它的属性。
AM的属性：

```
 amname |     name      | pg_indexam_has_property
--------+---------------+-------------------------
 rum    | can_order     | f
 rum    | can_unique    | f
 rum    | can_multi_col | t
 rum    | can_exclude   | t -- f for gin
```

索引层面的属性：

```
     name      | pg_index_has_property
---------------+-----------------------
 clusterable   | f
 index_scan    | t -- f for gin
 bitmap_scan   | t
 backward_scan | f
```

注意，与GIN索引不同的是，RUM支持Index scan —— 否则它就不可能返回LIMIT算子的结果。它不需要使用gin_fuzzy_search_limit参数。它可以支持排它约束。

列层面的约束：

```
        name        | pg_index_column_has_property
--------------------+------------------------------
 asc                | f
 desc               | f
 nulls_first        | f
 nulls_last         | f
 orderable          | f
 distance_orderable | t -- f for gin
 returnable         | f
 search_array       | f
 search_nulls       | f
```

RUM支持排序操作符。但是，这不是对所有操作符都成立：例如，对tsquery_ops就不成立。

英文资料翻译
(Https://Www.Mengqingzhong.Com/Category/%E8%8b%B1%E6%96%87%E8%B5%84%E6%96%99%E7%Bf%Bb%E8%Af%91/)