
Boris Muratshin @zzeng
User
91.0 karma 57.6 rating

23 Publications | 128 Comments | 37 Favorites | 34 Followers

Profile

January 9 at 08:05

Working → Pro and Z-order R-tree

Geoinformation Services * algorithms *, the PostgreSQL *, the C *

The index based on Z-order curve in comparison with R-tree has many advantages, it:

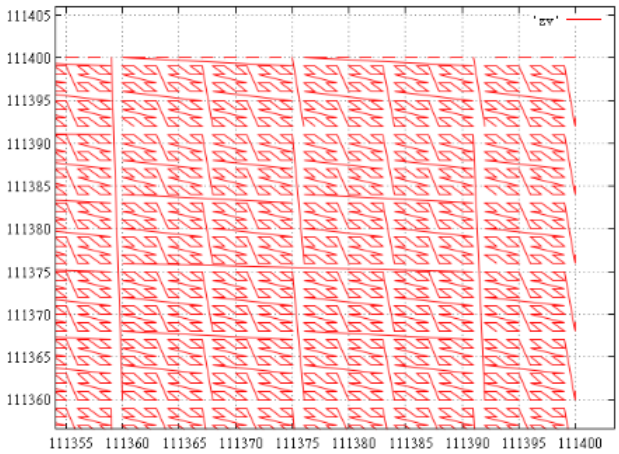
- implemented as a conventional B-tree, and we know that
- page B-tree have the best occupancy rate, in addition,
- Z-keys themselves are more compact
- B-tree traversal order has a natural, unlike R-tree
- B-tree faster builds
- B-tree is better balanced
- B-tree clear, it does not depend on heuristics splitting / merging pages
- B-tree does not degrade under constant change
- ...

However, the indices on the basis of Z-order may have the drawback - the relatively low productivity :). Under the cut, we try to understand what is involved in this deficiency and can we do something about it.

Roughly, the meaning of Z-curve is as follows - we alternate bits x & y coordinates in a single value, as a result. Example - (x = 33 **(010 0001),** y = 22 *(0110 01),* Z = ***1577 (0 1 1 0 0 0*** 1 0 1 0 0 1) If the coordinates of two points are close, then most likely, and Z-. values they will be close.
Three-dimensional (or more) variant has the same structure, we alternate level three (or more) coordinate.

And to find in some of the extents we have "rasterize" all the values of Z-curve to this extent, finding continuous intervals and in each form to search.

Here is an illustration of the extent [111000 ... 111000] [111400 ... 111400] (675 slots), the upper right corner (every continuous polygonal - single spaced):



And, for example, to the extent [111000 ... 111000] [112000 ... 112000] we get 1,688 continuous intervals is obviously their number essentially depends on the length of the perimeter. Looking ahead, on the test data to this extent it has got 15 points in 6 intervals.

Yes, most of these small intervals, up to degenerate - one value. Nevertheless, even if this Extent around only one value, it can be in any of the intervals. And, whether we like it or not, will have to do in 1688 subqueries to find out how many points actually.

View all of the value of the lower-left corner to the upper right - not an option, the difference between the angles in this case - 3144768, we have to look at more than three times as much data, and this is not the worst case. For example, the extent [499500 ... 499500] [500500 ... 500500] will range in value 135 263 808, in ~ 135 times the area of extent.

And then we can ask the traditional question -

What if ...

Suppose we do an empty index, really, we should do all these hundreds and thousands of sub-queries, to understand it? No, only one - from the lower left to the upper right corner.

Now suppose that extent is quite small, and the data is sparse and chances of finding something small. Perform the same query from corner to corner. If nothing is found, then it did not. Otherwise, there is a chance. But as we have seen, the area swept out by request from corner to corner can repeatedly exceed the extent of the search, and we have no desire to read clearly unnecessary data. Therefore, we will not see the entire mouse, and take from it only the minimum Z-value. To do this, the query is executed with the (order by), and (top 1).

So, we have a certain value. Suppose this value is not of our extent that it can give? It's time to remember that we have a sorted array [1 ... N] subqueries ranges. We will perform a binary search and find out what subqueries squeezed between this value, though, between m and m + 1. Great, so requests from 1 to m, you can not do, there is certainly nothing.

If the value belongs to our extents it means falls into one of our ranges and we can also be found in some, albeit too m. As before, the requests numbered 1 ... m-1 can be omitted. But the interval with m number deserves a separate query that will give us all that it is located.

What may still have the data, we will continue. Again fulfill the request, but not from corner to corner, and from the beginning of the interval m + 1 to the upper right corner. And we will do so, until we reach the end of the interval list.

That's the whole idea, note, it works fine and when the extent of the falls in a lot or a lot of data. At first glance, this would drastically reduce the number of requests and speed up the work.

It's time to test the idea in practice. As a test platform become used PostgeSQL 9.6.1, GiST . Measurements were carried out on a modest virtual machine with two cores and 4 GB of RAM, so the times do not have an absolute value, and this is the number of pages read can trust.

Initial data

In 100 million random points in the extent used as data of [0 ... 1000000] [0 ... 1000000].

Head table for 2-dimensional point data:

```
create table test_points (x integer,y integer);
```

These will create:

gawk script

Sort the resulting file (explained below) and fill it with a COPY statement in the table:

```
COPY test_points from '/home/.../data.csv';
```

Fill the table takes a few minutes. Data Size (\ dt +) - 4358 Mb

R-tree

The corresponding index is created using:

```
create index test_gist_idx on test_points using gist ((point(x,y)));
```

But there is a nuance. In a hash index is built for a long time (at least in the author he did not have to line up for the night). Building on pre-sorted data took about an hour.

Index size (\ di +) - 9031 Mb

In fact, the order of the data in the table is not important for us, but it must be shared by different methods, so I had to use a sorted table.

Test query looks like this:

```
select count(1) from test_points where point(x,y) <@ box(point("xmin","ymin"),point("xmax","ymax"));
```

Check for normal indices

To test the efficiency will perform spatial queries and individual indexes on x & y. They are made as follows:

```
create index x_test_points on test_points (x); create index y_test_points on test_points (y);
;
create index x_test_points on test_points (x); create index y_test_points on test_points (y);
```

It takes it a few minutes.

Test query:

```
select count(1) from test_points where x >= "xmin" and x <= "xmax" and y >= "ymin" and y <= "ymax";
```

Z-index

Now we need a function which is able to convert the x, y coordinates in the Z-value.

First, create an extension , there function:

```
CREATE FUNCTION zcurve_val_from_xy(bigint, bigint) RETURNS bigint AS 'MODULE_PATHNAME' LANGUAGE C IMMUTABLE STRICT;
```

```
RETURNS bigint
```

```
CREATE FUNCTION zcurve_val_from_xy(bigint, bigint) RETURNS bigint AS 'MODULE_PATHNAME' LANGUAGE C IMMUTABLE STRICT;
```

Her body:

Now (after the CREATE EXTENSION, of course) Z-index is constructed as follows:

```
create index zcurve_test_points on test_points(zcurve_val_from_xy(x, y));
```

This takes a few minutes (and does not require sorting data).

the index size (\ di +) - 2,142 Mb (~ 4 times less than that of R-tree)

Z-search index

So, in our first (let's call it "naive") version, we will do so:

1. To the extent dx * dy sized factory identifier array of the appropriate size
2. For each point in the extent vychislyaes Z-value
3. Sort the array of IDs
4. We find the continuous intervals
5. For each type of subquery interval execute:

```
select * from test_points where zcurve_val_from_xy(x, y) between $1 and $2
```

6. We get results

For the search function will use (lower body) via this variant:

```
CREATE FUNCTION zcurve_oids_by_extent(bigint, bigint, bigint, bigint) RETURNS bigint AS 'MODULE_PATHNAME' LANGUAGE C IMMUTABLE STRICT;
```

```
bigint, bigint)
```

```
CREATE FUNCTION zcurve_oids_by_extent(bigint, bigint, bigint, bigint) RETURNS bigint AS 'MODULE_PATHNAME' LANGUAGE C IMMUTABLE STRICT;
```

Spatial query with this function looks like this:

```
select zcurve_oids_by_extent("xmin","ymin","xmax","ymax") as count;
```

This function returns only the number of hits, the data may optionally be removed using the " elog (INFO ...) ".

The second, improved, (let's call it "with samples") option is as follows:

1. to the extent dx * dy sized factory identifier array of the appropriate size
2. for each point in the extent vychislyaes Z-value
3. sort an array of IDs
4. We find continuous intervals
5. We start with the first found range:

1. We perform a "test" type of request (parameters - limit of the range):

```
select * from test_points where zcurve_val_from_xy(x, y) between $1 and $2 order by zcurve_val_from_xy(x, y) limit 1
```

2. This query gives us a table row with the smallest Z-value from the beginning of the test interval to the end of the search extent.
3. If nothing is found the request, then the search and data Extent left, we leave.
4. Now we can analyze the Z-value is found:

    1. Look, if it fell into some - any of our slots.
    2. If not, we find the next number of the remaining interval and go to step 5.
    3. If hit, fulfilling a request for this type of interval:

```
select * from test_points where zcurve_val_from_xy(x, y) between $1 and $2
```

    4. Take the next interval and go to step 5.

To search using this embodiment will use the function:

```
CREATE FUNCTION zcurve_oids_by_extent_ii(bigint, bigint, bigint, bigint) RETURNS bigint AS 'MODULE_PATHNAME' LANGUAGE C IMMUTABLE STRICT;
```

bigint, bigint)

```
CREATE FUNCTION zcurve_oids_by_extent_ii(bigint, bigint, bigint, bigint) RETURNS bigint AS 'MODULE_PATHNAME' LANGUAGE C IMMUTABLE STRICT;
```

Spatial query with this function looks like this:

```
select zcurve_oids_by_extent_ii("xmin","ymin","xmax","ymax") as count;
```

The function also returns only the number of hits.

rasterization

In the described algorithm to use a very simple and inefficient algorithm "rasterization" - producing intervals list.

On the other hand, it is easy to perform measurements of the average time of his work at random extents size. Here they are:

Extent dx * dy	The expected average number of points in the issue	Time, ms
100X100	1	.96 (.37 + .59)
316X316	10	11 (3.9 + 7.1)
1000X1000	100	119.7 (35 + 84.7)
3162X3162	1000	1298 (388 + 910)
10000X10000	10000	14696 (3883 + 10813)

In parentheses are indicated separately two phases - the calculation of Z-values and sorting

results

Here is a summary table with the data.

Npoints	Type	Time (ms)	Reads	Shared Hits
1	X & Y rtree	43.6 .5	59.0173 4.2314	6.1596 2.6565

	Z-value Z-value-ii	8.3 (9.4) 1.1 (2.2)	4.0988 4.1984 (12,623)	803,171 20.1893 (57,775)
10	X & Y rtree Z-value Z-value-ii	83.5 .6 15 (26) 4 (15)	182,592 13.7341 14,834 14,832 (31,439)	9.24363 2.72466 2527.56 61,814 (186,314)
100	X & Y rtree Z-value Z-value-ii	220 2.1 80 (200) 10 (130)	704,208 95.8179 95,215 96,198 (160,443)	16,528 5.3754 8007.3 208,214 (600,049)
1000	X & Y rtree Z-value Z-value-ii	740 12 500 (1800) 200 (1500)	3176.06 746,617 739.32 739.58 (912,631)	55,135 25,439 25816 842.88 (2028.81)
10000	X & Y rtree Z-value Z-value-ii	2500 70 ... 1200 4700 (19000) 1300 (16000)	12393.2 4385.64 4284.45 4305.78 (4669)	101.43 121.56 86274.9 5785.06 (9188)

Npoints - the average number of points in the issue.

Type -

- 'X & Y' - the use of separate indices for x & y
- 'Rtree' - request through the R-tree
- Z-value - an honest search intervals
- 'Z-value-ii' - search intervals with samples

Time (ms) - the average time of the request. Under these conditions, the value is very unstable, depends on the database cache and disk cache from the virtual machine, and from the disk cache of the host system. Here it is more likely to help. *For the Z-value and Z-value-ii shows the number 2. In brackets - the actual time. Without brackets - time net costs "rasterization".*

Reads - the average number of readings to a request (received via EXPLAIN (ANALYZE, BUFFERS))

Shared hits - the number of accesses to buffers (...)
For the Z-value-ii in columns Reads & Shared hits given number 2. In brackets - the total number of readings. Without brackets - minus probe requests to order by and limit 1. This was done because of the opinion that such a request is being proofread all the data in the interval, sorts and gives a minimum, rather than just give a value of 1 is already sorted index. Therefore, requests for such statistics was considered excessive, but shown for reference.

Times are shown in the second run, in the heated servers and virtual machines. The number of read buffers - on the freshly-lifted server.

In all types of requests were read and the data of the table itself does not belong to the indices. However, this same data to the same table, so that we have constant values for all the types of queries.

conclusions

1. R-tree is still very well in static pages readings efficiency is very high.
2. But the Z-order index is sometimes necessary to read the pages on which there is no relevant data. This occurs when the cursor enters the testing between the intervals, it is likely that in this interval will be many foreign points and a particular page contains no totals.
3. **Nevertheless, due to the denser packing Z-order close to the R-tree on the actual number of pages read.** This suggests that the Z-order **is potentially** capable of delivering similar performance.
4. Z-order index reads a huge number of pages from the cache because repeatedly made requests to the same place. On the other hand, these readings are relatively inexpensive.
5. For large Z-order requests loses much speed R-tree. This is explained by the fact that for the implementation of sub-queries, we use the SPI - a high-level and not too fast mechanism. And with "rasterization", of course, we must do something.
6. At first glance, the use of sampling intervals is not much and accelerated work, and formally even worsened Statistics page read. But we must understand that it is high-cost funds, which had to use. Potentially, the index on the basis of Z-order is not worse than the R-tree performance and much better for the rest of the tactical and technical characteristics.

prospects

To create on the basis of Z-order full spatial index curve, which would be able to compete on equal terms with the R-tree, it is necessary to solve the following problems:

- to come up with an inexpensive algorithm to obtain a list of sub-intervals for the extent at
- switch to low-level access to the tree of the index

Fortunately, both does not seem something impossible.

source

Go on here, not posted on github as code is purely experimental and has no practical value.

PPS: Many thanks to the guys from PostgresPro for What inspired me for this work.

the postgresql , index the spatial , tree-r , ZOrder , spi

33 6,8k 79

Boris Muratshin @zzeng karma 91.0 rating 57.6
User

Related publications

31 Transaction isolation levels with examples in the PostgreSQL
13k 247 18

27 Optimizing a query with GROUP BY in the PostgreSQL
8,6k 80 37

15 PostgreSQL slave + btrfs and systemd = hot test base
4,8k 84 34

Most popular

Development

Now **Day** **A week** **Month**

32 Why I do not like synthetic benchmarks
1,8k 8 18

22 ROC has launched the first Orthodox messenger
35,5k 28 169

15 Visual C ++ for Linux Development: The practice of using for Windows developers
2,3k 30 9

15 Digest of fresh materials from the world for last week frontend №248 (January 30 - February 5, 2017)
7,1k 58 4

13 Electronic ink Wirenboard 5 or draw bar codes on the Go
1,1k 12 0

Comments (7)

neolink January 10, 2017 at 02:32 #

+1

For completeness, it is worth to check the decision on the forehead with a composite index

```
create index test_points_x_y on test_points (x, y);
```

 **zzeng** January 10, 2017 at 10:49 # 卜 ↑ +1

looked with one eye,
a composite index takes 2GB
query select count (1) ... is fulfilled as the index only scan, ie without referring to a table read from pages 2 to 5 times less in comparison with X & Y.
But it is rather a good example of high-quality optimizer than the merit of the actual index.

 **kashey** January 10, 2017 at 12:47 # 2

Very good, but there are a couple of problems:
1. the Z-below code is actually linear quad-tree. Not R. B. And it is not all that you have stated bonuses at the beginning of the article - does not work. No balancing there. And there is no tree. It's a cross between a fractal curve and - spatial filling curve, and their is a lot different. The meaning of one - reduced dimension.
2. Try to Hilbert. Consider it more difficult, but more "similar" values in it located "closer"
3. "The Secret" work Z-value-ii is well described in the internet as a BIGMIN / LITMAX (https://en.wikipedia.org/wiki/Z- order_curve # Use_with_one-dimensional_data_structures_for_range_searching)

In general, highly recommend an active friendship with this math. It really works well, but we need to understand the limitations.
And do not forget - Z is the point. While the R-tree continued to eat a cactus.

> PS: Even in the Internet, you can find a variety of functions faster construction of Z-code
> PPS: Some versions of R-tree built on their Z
> PPPS: WELCOME se the Club!

 **zzeng** January 10, 2017 at 13:31 # 卜 ↑ +1

according to claim 1 Z-code - the number and the numbers in this case, we laid in the B-tree.
The fact that you (apparently) had in mind, called kd-trees and it's a different story.
And do not forget - this is the point Z, a rectangle - the four-point Z.

 **kashey** January 10, 2017 at 13:59 *(Comment has been changed)* # 卜 ↑ 2

1. In principle, yes, on the side of the database is to meet the B-tree. It turns out quite honestly. Take a look at it without the database.
2. By the way - Z-code, which is sometimes called «bit interleaving», in principle, KD. Every bit of space cut its plane.
3. Only here the search in the vicinity of a four-dimensional point is a little difficult :)

In general, Z (and the company) simply allows the beginning somehow put in a certain interval of the data, and then look for it.
If this is the "fast" search for codes when you need a specific "cell" virtual quadtree, is best (technically) works through a mask. You say why the code should start - commit first dipping step into a tree, and after a certain time the "release" the code, allowing it to constantly changing freely.
If all code 4 bits are Level 2, the fixation 0b1000 0b0111 mask matches all elements in the left (or right) subtree, those between 0b1000 and 0b1111.
Basically - it is - the ability to use these "numbers" as NestedSet - one of the foundations of a feature of these curves. Over the past years for different spatial filling came up with a lot of drug addiction. Textures in video memory on them laid. Gray Code, on which 99% of electronics works - also spatial filling.

To my regret - I'm not the best storyteller, but perhaps one old the article is a little help to understand what I meant.

 **zzeng** January 10, 2017 at 14:09 # 卜 ↑ 2

Thanks for the link, in principle, I am aware of some of this magic.
And this article discusses how to describe the introduction of the search algorithm is on the B-tree, as the most honest way to store data in one-dimensional (by and large) the one-dimensional disk space.
A week later - another welcome.
And one old little article there will not only have you :)

 **kashey** January 10, 2017 at 14:21 # 卜 ↑ +1

ABOUT!Monsieur knows his sausage scraps. I shake your hand, a colleague.

Sport is dead! (based on the «Agile died" and other obituaries) 4

RAS recognized homeopathy pseudoscience 55

What is Traffic Arbitrage? 1

Why I do not like synthetic benchmarks 16

The story of the designer, endearing mathematics 1