Institute of RS and GIS, Peking University

GiST Build with Pre-sorting Methods

Han Wang
hanwgeek@gmail.com

北京大學
PEKING UNIVERSITY

Contents

PRELIMINARIES

Basic Concepts of GiST

Definition

GiST(Generalized Search Tree) is a generalization data structure of a variety of disk-based height-balanced search trees.

Structure

- A balanced tree of variable fanout between kM and M, $\frac{2}{M} \leq k \leq \frac{1}{2}$
- p is predicate, ptr is pointer to tuples
- Non-Leaf/Leaf node: (p, ptr)

Features

- Non-leaf node: p is true when instantiated with the values of any tuple reachable from ptr
- Leaf node: p is true when instantiated with values from the indicated tuple

Key/Tree Methods of GiST

- p: a predicate q: a query predicate
- E: an entry E = (p;ptr) P: a set of entries $\{E_1 = (p_1, ptr_1), E_2 = (p_2, ptr_2), \ldots\}$

Methods

- Consistent(E, q): returns false if $p \cap q$ can be guaranteed unsatisfiable
- Union(P): returns some predicate r that holds for all tuples stored
- Penalty(E_1, E_2): returns a domain-specific penalty for inserting $E2$ into the subtree rooted at $E1$
- PickSplit(P): given a set P of $M + 1$ entries, splits P into two sets of entries $P1$, $P2$
- ...

- Search(R, q): Search all tuples that satisfy q from root R
- Insert(R, E, l): new GiST resulting from insert of E at level l from root R

Applications

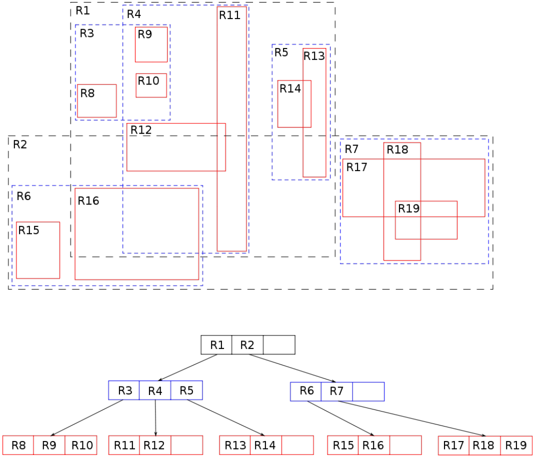

Figure: R-tree(Wikipedia)

- B-tree
- B+-tree
- R-tree
- hB-tree
- RD-tree
- ...

IMPLEMENTATION

GiST Index Building in Postgres

In PostgreSQL:

Building Strategies

1. Start with an empty index, and insert all tuples one by one.
2. Sort all input tuples, pack them into GiST leaf pages in the sorted order, and create downlinks and internal pages as we go. This builds the index from the bottom up, similar to how B-tree index build.
   (With SortSupport API provided)

It is obvious that we have to define an "order" for the tuples to sort them in advance

Index for Geometry Objects in PostGIS

In PostGIS:

BOX2DF Structure

```
CREATE OPERATOR CLASS gist_geometry_ops_2d
    DEFAULT FOR TYPE geometry USING GIST AS
    STORAGE box2df
    OPERATOR     1     <<  ,
    ...
    FUNCTION     11    geometry_gist_sortsupport_2d(internal);
```

Sort support function

```
CREATE OR REPLACE FUNCTION geometry_gist_sortsupport_2d(internal)
    RETURNS internal
    AS '$libdir/postgis-3', 'gserialized_gist_sortsupport_2d'
    LANGUAGE 'c' PARALLEL SAFE
    COST 1;
```

Sort Support Function API

```
Datum gserialized_gist_sortsupport_2d(PG_FUNCTION_ARGS) {
    SortSupport ssup = (SortSupport) PG_GETARG_POINTER(0);

    if (ssup->abbreviate)
    {
        ssup->comparator = hash_cmp;
        ssup->abbrev_converter = hash_abbrev_convert;
        ssup->abbrev_abort = hash_abbrev_abort;
        ssup->abbrev_full_comparator = hash_abbrev_full_cmp;
    }
    else
    {
        ssup->comparator = hash_abbrev_full_cmp;
    }
    PG_RETURN_VOID();
}
```

Sort Support Function API

```
static int hash_cmp(Datum a, Datum b, SortSupport ssup) {
        if (a > b) return 1;
        else if (a < b) return -1;
        else return 0;
}

static Datum hash_abbrev_convert(Datum original, SortSupport ssup) {
        BOX2DF *box = (BOX2DF *)original;
        union floatuint {
                uint32_t u;
                float f;
        };

        union floatuint x, y;
        x.f = (box->xmax + box->xmin) / 2;
        y.f = (box->ymax + box->ymin) / 2;
        return (Datum)uint32_hilbert(y.u, x.u);
}
```

Order of Geometry Objects

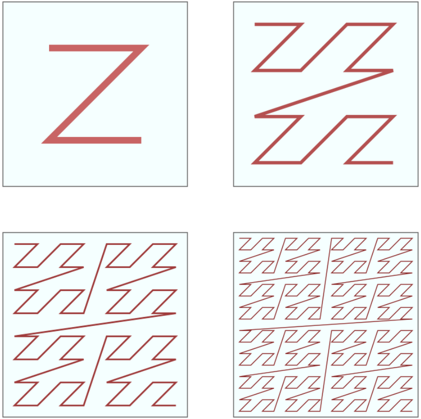

Figure: Z-order(Wikipedia)

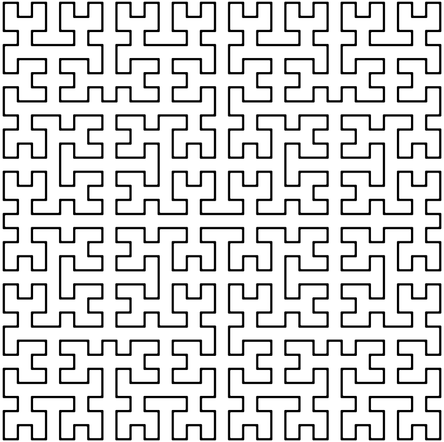

Figure: Hilbert Curve(Squircular)

Order of Geometry Objects

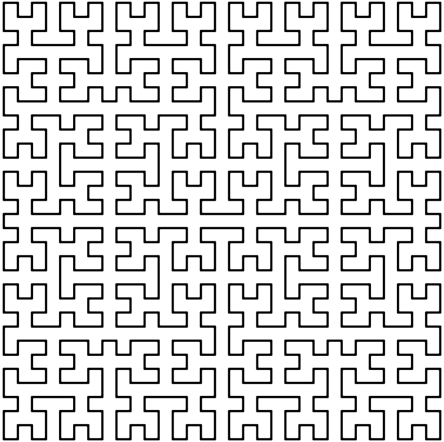

Figure: Hilbert Curve(Squircular)

- Infinite subdivision and approximation
- Maintain the proximity that exists in high dimensions in the 1-d case

Fast Hash Function

Given a $d * n$-bit number, split the index into n groups i_j of d bits each

$$
\begin{aligned}
&x_0 = q(i_0) = T_0 * q(i_0) \\
&x_1 = t(i_0) * q(i_1) = T_1 * q(i_1) \\
&x_2 = t(i_0) * t(i_1) * q(i_2) = T_2 * q(i_2) \\
&\ldots \\
&x_{n-1} = t(i_0) * t(i_1) * \ldots * t(i_{n-2}) * q(i_{n-1}) = T_{n-1} * q(i_{n-1})
\end{aligned}
$$

- q: Function mapping d index bits to an orthant
- t: Function mapping d index bits to an element of the transformation group
- $*$: The operator of that group
- Apply bit-wise opration like:

```
A = ((a & (a >> 4)) ^ (b & (b >> 4)));
B = ((a & (b >> 4)) ^ (b & ((a ^ b) >> 4)));
C ^= ((a & (c >> 4)) ^ (b & (d >> 4)));
D ^= ((b & (c >> 4)) ^ ((a ^ b) & (d >> 4)));
```

http://threadlocalmutex.com/?p=126

PERFORMANCE TEST

Performance Test

Search a small patch in a data area:

Index	Building Time (ms)	Plan Time (ms)	Buffer Hit Number	Excution Time (ms)
No index	0	0.05	834	13.1
Default GiST	450	0.05	12	0.016
Z-order Pre-sort GiST	130	0.05	15	0.046
Hilbert Pre-sort GiST	140	0.05	15	0.047

Traverse the data area with a small patch(Mean):

Index	Building Time (ms)	Plan Time (ms)	Buffer Hit Number	Excution Time (ms)
No index	0	0.042	834	12.96
Default GiST	450	0.057	13.52	0.016
Morton Pre-sort GiST	130	0.049	16.98	0.055
Hilbert Pre-sort GiST	140	0.050	15.37	0.046

CONCLUSION

Conclusion

Conclusion

- Space filling curve hash function does improve the index building performance
- But pre-sort index with hash functions leads to query performance loss

What's Next

- To improve the query performance with an optimized hash function
- To implement a n-dimensional hash function

THANKS

References

- Hellerstein, J. et al. Generalized Search Trees for Database Systems. VLDB (1995).
- PostgreSQL GiST document: https://www.postgresql.org/docs/14/gist.html
- Postgres SortSupport: https://brandur.org/sortsupport
- Hilbert Curve Packing: https://observablehq.com/@mourner/hilbert-curve-packing
- Fast Hilbert Hash Implementation: http://threadlocalmutex.com/?p=126