Directing PostgreSQL to Performance

PostgreSQL Index improvements over time

Matthias van de Meent

PostgreSQL hacker/contributor since 2020

Software Engineer at Neon

NOTE:

This presentation will show you some indexes that have no basis in reality. Do not try to reproduce these in production systems.

What am I covering?

- Included Index AMs on a high level
- Existing, planned and potential future improvements
- Not covering:
    - Profound knowledge of Index AM X
    - Selling Index AM X
    - The existing uses of Index AM X for your database

Refresher

Why do we need indexes?

1. Performance of finding one row by T.uuid is O(tablesize)
2. Add hash index on T.uuid
3. Performance of finding one row by T.uuid is now ~ O(1)

- Improve query times for common access patterns

How do indexes work?

- Table storage is an unorganized HEAP
  - *CREATE TABLE ... USING heap*
- IO is expensive
- Use the least amount of block accesses to get to your result
  - Inclusion (result is *somewhere* in there)
  - Exclusion (result is *definitely not* in there)

How do btree indexes work?

- Ordered, tree-structured index
  - Ordered by index key
  - Leaf entries point to heap tuples
- Fan-out of 300+ is common
  - Low tree depth thus few blocks accessed to find value

How do hash indexes work?

- Hash table
- Can only do equality checks
- Relatively small size, good for point lookups

How do GiST indexes work?

- Tree-structured index
  - Excludes downlinks when not 'consistent'
- Any balanced tree structure: GiST = Generalized Search Tree

How do GIN indexes work?

- Deformed keys
- 'tree of trees'

How do BRIN indexes work?

- Summarized results for key columns
- O(tablesize) index scan
  - BUT: Order(s) of magnitude smaller
- Built to exclude large ranges of data, fast

Important distinctions

- Index size
- Index bloat
  - Tuples: Index contains tuples that point to now-invisible tuples
  - Space: Index uses more pages than strictly necessary

What has improved?

CREATE INDEX

- Pre-sorting
  - GiST, hash
- Sorting infrastructure
  - All pre-sorted index builds

Has improved: Index size

- Strict tuple ordering ⇒ suffix truncation
  - btree (Anastasia Lubennikova, Peter Geoghegan)
- Deduplication
  - btree (Peter Geoghegan, Heikki Linnakangas)

Suffix truncation

offno (4B)	t_tid (6B)	flags + size (2B)	index tuple data	tuple size
0	(1, 1)	…	NULL, 1	28.00
1	(1, 2)	…	NULL, 1	28.00
2	(1, 3)	…	NULL, 1	28.00
3	(1, 3)	…	NULL, 1	28.00
…	…	…	…	…
34	(1, 34)	…	NULL, 4	28.00
			space used	980.00
			TIDs	35.00
			bytes/entry	28.00

BLCKSZ=1kB

Has improved: Index size

- Strict tuple ordering ⇒ suffix truncation
  - btree (Anastasia Lubennikova, Peter Geoghegan)
- Deduplication
  - btree (Peter Geoghegan, Heikki Linnakangas)

Full btree leaf page

offno (4B)	t_tid (6B)	flags + size (2B)	index tuple data	tuple size
0	(1, 1)	…	NULL, 1	28.00
1	(1, 2)	…	NULL, 1	28.00
2	(1, 3)	…	NULL, 1	28.00
3	(1, 3)	…	NULL, 1	28.00
…	…	…	…	…
34	(1, 34)	…	NULL, 4	28.00
			space used	980.00
			TIDs	35.00
			bytes/entry	28.00

BLCKSZ=1kB

Full btree leaf page + deduplication

offno (4B)	t_tid (6B)	flags + size (2B)	index tuple data	tuple size
0	(0, BT_IS_POSTING | 10)	INDEX_AM_RESERVED_BIT	NULL, 1, tid[]{(1, 1), (1, 2), … (1, 10)}	92.00
1	(0, BT_IS_POSTING | 10)	INDEX_AM_RESERVED_BIT	NULL, 2, tid[]{(1, 11), (1, 12), … (1, 20)}	92.00
2	(0, BT_IS_POSTING | 10)	INDEX_AM_RESERVED_BIT	NULL, 3, tid[]{(1, 21), (1, 22), … (1, 30)}	92.00
…	…	…	…	…
9	(0, BT_IS_POSTING | 10)	INDEX_AM_RESERVED_BIT	NULL, 10, tid[]{(1, 91), (1, 92), … (1, 100)}	92.00
			space used	920.00
			TIDs	100.00
			bytes/entry	9.20

BLCKSZ=1kB

Has improved: Index bloat

- Bottom-up index deletion
  - btree (PG14, Peter Geoghegan)

Index bloat:
- Tuples in the index that are invisible to any transaction
- More space used by the index than necessary

What is being improved?

Index creation

- Improving efficacy of pre-sorts in Hash
    - order by (bucket, hash) instead of only (hash)

VACUUM performance

- new HOTness with BRIN
  - We store no TIDs in BRIN, so there is no need to break HOT for BRIN (~~PG15~~ PG16? Josef Simanek, Tomas Vondra)
- heapam
  - Compacter, more efficient dead tuple storage (Masahiko Sawada)

Index performance

- nbtree: dynamic prefix compression

Rudimentary btree index search

Search for first row < (1, 1, 2, 4)

TID	Column 1	Column 2	Column 3	Column 4
1	1	1	1	1
2	1	1	1	1
3	1	1	1	2
4	1	1	1	2
5	1	1	2	3
6	1	1	2	3
7	1	1	2	4
8	1	1	2	4
9	1	2	3	5
10	1	2	3	5
11	1	2	3	6
12	1	2	3	6
13	1	2	4	7
14	1	2	4	7
15	1	2	4	8
16	1	2	4	8
17	2	3	5	9
18	2	3	5	9
19	2	3	5	10
20	2	3	5	10
21	2	3	6	11
22	2	3	6	11

Rudimentary btree index search

Search for first row < (1, 1, 2, 4):
- < (1, 2, ...)

TID	Column 1	Column 2	Column 3	Column 4
1	1	1	1	1
2	1	1	1	1
3	1	1	1	2
4	1	1	1	2
5	1	1	2	3
6	1	1	2	3
7	1	1	2	4
8	1	1	2	4
9	1	2	3	5
10	1	2	3	5
11	1	2	3	6
12	1	2	3	6
13	1	2	4	7
14	1	2	4	7
15	1	2	4	8
16	1	2	4	8
17	2	3	5	9
18	2	3	5	9
19	2	3	5	10
20	2	3	5	10
21	2	3	6	11
22	2	3	6	11

Rudimentary btree index search

Search for first row < (1, 1, 2, 4):
< (1, 2, ...)
> (1, 1, 2, 3)

TID	Column 1	Column 2	Column 3	Column 4
1	1	1	1	1
2	1	1	1	1
3	1	1	1	2
4	1	1	1	2
5	1	1	2	3
6	1	1	2	3
7	1	1	2	4
8	1	1	2	4
9	1	2	3	5
10	1	2	3	5
11	1	2	3	6
12	1	2	3	6
13	1	2	4	7
14	1	2	4	7
15	1	2	4	8
16	1	2	4	8
17	2	3	5	9
18	2	3	5	9
19	2	3	5	10
20	2	3	5	10
21	2	3	6	11
22	2	3	6	11

Rudimentary btree index search

Search for first row < (1, 1, 2, 4):
< (1, 2, ...)
> (1, 1, 2, 3)
< (1, 1, 2, 4)

TID	Column 1	Column 2	Column 3	Column 4
1	1	1	1	1
2	1	1	1	1
3	1	1	1	2
4	1	1	1	2
5	1	1	2	3
6	1	1	2	3
7	1	1	2	4
8	1	1	2	4
9	1	2	3	5
10	1	2	3	5
11	1	2	3	6
12	1	2	3	6
13	1	2	4	7
14	1	2	4	7
15	1	2	4	8
16	1	2	4	8
17	2	3	5	9
18	2	3	5	9
19	2	3	5	10
20	2	3	5	10
21	2	3	6	11
22	2	3	6	11

Rudimentary btree index search

Search for first row < (1, 1, 2, 4):
< (1, 2, ...)
> (1, 1, 2, 3)
< (1, 1, 2, 4)
< (1, 1, 2, 4)

TID	Column 1	Column 2	Column 3	Column 4
1	1	1	1	1
2	1	1	1	1
3	1	1	1	2
4	1	1	1	2
5	1	1	2	3
6	1	1	2	3
7	1	1	2	4
8	1	1	2	4
9	1	2	3	5
10	1	2	3	5
11	1	2	3	6
12	1	2	3	6
13	1	2	4	7
14	1	2	4	7
15	1	2	4	8
16	1	2	4	8
17	2	3	5	9
18	2	3	5	9
19	2	3	5	10
20	2	3	5	10
21	2	3	6	11
22	2	3	6	11

Rudimentary btree index search

Search for first row < (1, 1, 2, 4):
< (1, 2, ...)
> (1, 1, 2, 3)
< (1, 1, 2, 4)
< (1, 1, 2, 4)
> (1, 1, 2, 3)

TID	Column 1	Column 2	Column 3	Column 4
1	1	1	1	1
2	1	1	1	1
3	1	1	1	2
4	1	1	1	2
5	1	1	2	3
6	1	1	2	3
7	1	1	2	4
8	1	1	2	4
9	1	2	3	5
10	1	2	3	5
11	1	2	3	6
12	1	2	3	6
13	1	2	4	7
14	1	2	4	7
15	1	2	4	8
16	1	2	4	8
17	2	3	5	9
18	2	3	5	9
19	2	3	5	10
20	2	3	5	10
21	2	3	6	11
22	2	3	6	11

Rudimentary btree index search

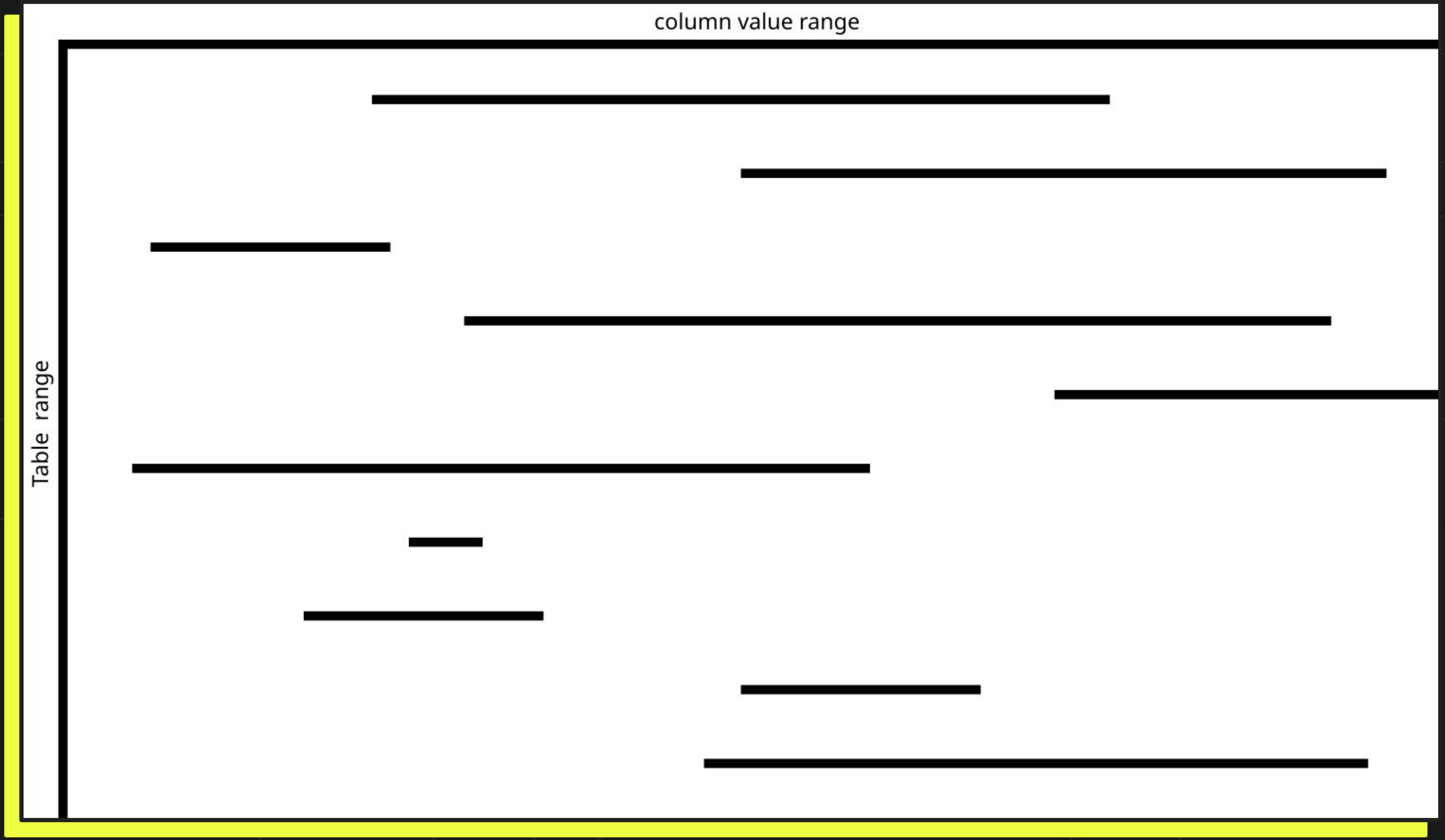
Search for first row < (1, 1, 2, 4):
< (1, 2, ...)
> (1, 1, 2, 3)
< (1, 1, 2, 4)
< (1, 1, 2, 4)
> (1, 1, 2, 3)

TID	Column 1	Column 2	Column 3	Column 4
1	1	1	1	1
2	1	1	1	1
3	1	1	1	2
4	1	1	1	2
5	1	1	2	3
6	1	1	2	3
7	1	1	2	4
8	1	1	2	4
9	1	2	3	5
10	1	2	3	5
11	1	2	3	6
12	1	2	3	6
13	1	2	4	7
14	1	2	4	7
15	1	2	4	8
16	1	2	4	8
17	2	3	5	9
18	2	3	5	9
19	2	3	5	10
20	2	3	5	10
21	2	3	6	11
22	2	3	6	11

Index performance

- BRIN minmax-assisted table sort
  - Patch is currently under development (Tomas Vondra)

BRIN view of table



BRIN view of table

What could be improved?

ORDER BY support

- btree_gist:
  - supports ORDER BY myintcol <-> INT_MIN, ...
  - ... but not ORDER BY myintcol

Limiting index bloat

- Apply page split prevention (c.q. nbtree in PG14) in other trees:
  - GIST
  - SP-GiST
  - GIN

Index size

- btree
  - static, on-page prefix truncation
  - highkey truncation support from opclass
  - key normalization
- GIST prefix and suffix truncation
  - mostly in case of multi-column ordered opclasses

Full btree leaf page

offno (4B)	t_tid (6B)	flags + size (2B)	index tuple data	tuple size
0	(1, 1)	…	NULL, 1	28.00
1	(1, 2)	…	NULL, 1	28.00
2	(1, 3)	…	NULL, 1	28.00
3	(1, 3)	…	NULL, 1	28.00
…	…	…	…	…
34	(1, 34)	…	NULL, 4	28.00
			space used	980.00
			TIDs	35.00
			bytes/entry	28.00

BLCKSZ=1kB

Full btree leaf page + static prefix truncation

offno (4B)	t_tid (6B)	flags + size (2B)	index tuple data	tuple size
0	([7,16], BT_IS_PREFIX | 1)	INDEX_AM_RESERVED_BIT	NULL, 1	28.00
1	([17,26], BT_IS_PREFIX | 1)	INDEX_AM_RESERVED_BIT	NULL, 2	28.00
2	([27,36], BT_IS_PREFIX | 1)	INDEX_AM_RESERVED_BIT	NULL, 3	28.00
…	…	…	…	…
6	([67,76], BT_IS_PREFIX | 1)	INDEX_AM_RESERVED_BIT	NULL, 7	28.00
7	(1, 1)	…	()	12.00
8	(1, 2)	…	()	12.00
9	(1, 3)	…	()	12.00
…	…	…	…	…
73	(1, 66)	…	()	12.00
			space used	1,000.00
			TIDs	66.00
			bytes/entry	15.15

BLCKSZ=1kB

Full btree leaf page + dedup + static prefix truncation

offno (4B)	t_tid (6B)	flags + size (2B)	index tuple data	tuple size
0	([1,10], BT_IS_PREFIX | 1)	INDEX_AM_RESERVED_BIT	NULL	20.00
1	(0, BT_IS_POSTING | 10)	INDEX_AM_RESERVED_BIT	1, tid[]{(1, 1), (1, 2), … (1, 10)}	84.00
2	(0, BT_IS_POSTING | 10)	INDEX_AM_RESERVED_BIT	2, tid[]{(1, 11), (1, 12), … (1, 20)}	84.00
3	(0, BT_IS_POSTING | 10)	INDEX_AM_RESERVED_BIT	3, tid[]{(1, 21), (1, 22), … (1, 30)}	84.00
…	…	…	…	…
11	(0, BT_IS_POSTING | 10)	INDEX_AM_RESERVED_BIT	11, tid[]{(1, 101), (1, 102), … (1, 110)}	84.00
			space used	944.00
			TIDs	110.00
			bytes/entry	8.58

BLCKSZ=1kB

NEON

Thank you!

matthias@neon.tech