Comparison of SIMC Indexing Methods

(Written By Moyao Wang z5099162 and Shan Wang z5119666)

Ideas on comparison:

We are going to provide some test cases to directly display the difference of efficiency. And then we will also use the theory to verify the conclusion.

Test case 1:

Description: This test case is based on relation R, with 4 attributes, 5000 expected tuples, 2000 actual inserted tuples and 1/1000 false match probably.

- First, create a relation R and insert 2000 tuples into it.

```
./create R 5000 4 1000
./gendata 2000 4 | ./insert R
```

- Select with open query (?,?,?,?)

The output of selecting everything from relation R with different indexing method

```
$  time ./select R  ?,?,?,? x
Query Stats:
# signatures read:   0
# sig pages read:    0
# tuples examined:   2000
# data pages read:   21
# false match pages: 0

real        0m0.004s
user        0m0.000s
sys         0m0.000s
```

```
$ time ./select R  ?,?,?,? t
Query Stats:
# signatures read:   2000
# sig pages read:    4
# tuples examined:   2000
# data pages read:   21
# false match pages: 0

real        0m0.007s
user        0m0.004s
sys         0m0.000s
```

```
$ time ./select R  ?,?,?,? p
Query Stats:
# signatures read:   21
# sig pages read:    5
# tuples examined:   2000
# data pages read:   21
# false match pages: 0

real        0m0.006s
user        0m0.004s
sys         0m0.000s
```

```
$ time ./select R  ?,?,?,? b
Query Stats:
# signatures read:   0
# sig pages read:    10
# tuples examined:   2000
# data pages read:   21
# false match pages: 0

real        0m0.004s
user        0m0.000s
sys         0m0.000s
```

According to the output with different method, the running time of using bit-sliced signature as index is the fastest. Running time of using tuple and page signature are slower than the other one.

- Select with querying one solution (1001998,?,?,?)

The output of selecting from relation R using id=1001998 with different indexing method

```
$ time ./select
R  1001998,?,?,? x
Query Stats:
# signatures read:   0
# sig pages read:    0
# tuples examined:   2000
# data pages read:   21
# false match pages: 20

real        0m0.004s
user        0m0.000s
sys         0m0.000s
```

```
$ time ./select
R  1001998,?,?,? t
Query Stats:
# signatures read:   2000
# sig pages read:    4
# tuples examined:   157
# data pages read:   2
# false match pages: 1

real        0m0.003s
user        0m0.000s
sys         0m0.000s
```

```
$ time ./select
R  1001998,?,?,? p
Query Stats:
# signatures read:   21
# sig pages read:    5
# tuples examined:   60
# data pages read:   1
# false match pages: 0

real        0m0.003s
user        0m0.000s
sys         0m0.000s
```

```
$ time ./select
R  1001998,?,?,? b
Query Stats:
# signatures read:   9
# sig pages read:    10
# tuples examined:   60
# data pages read:   1
# false match pages: 0

real        0m0.002s
user        0m0.000s
sys         0m0.000s
```

According to the output with different method, the running time of using bit-sliced signature as index is the fastest. Running time of the others are basically the same.

- Select with querying many solution (?,?,a3-001,?)

The output of selecting from relation R using a3-001 with different indexing method

```
$ time ./select R ?,?,a3-
001,? x
Query Stats:
# signatures read:   0
# sig pages read:    0
# tuples examined:   2000
# data pages read:   21
# false match pages: 12

real        0m0.005s
user        0m0.000s
sys         0m0.000s
```

```
$ time ./select R ?,?,a3-
001,? t
Query Stats:
# signatures read:   2000
# sig pages read:    4
# tuples examined:   933
# data pages read:   10
# false match pages: 1

real        0m0.005s
user        0m0.000s
sys         0m0.000s
```

```
$ time ./select R ?,?,a3-
001,? p
Query Stats:
# signatures read:   21
# sig pages read:    5
# tuples examined:   836
# data pages read:   9
# false match pages: 0

real        0m0.004s
user        0m0.000s
sys         0m0.000s
```

```
$ time ./select R ?,?,a3-
001,? b
Query Stats:
# signatures read:   9
# sig pages read:    10
# tuples examined:   836
# data pages read:   9
# false match pages: 0

real        0m0.003s
user        0m0.000s
sys         0m0.004s
```

According to the output with different method, the running time of using bit-sliced signature as index is a little bit faster than using page signature. Running time of the others are slower and basically the same.

- Select with querying multiple values (1000001,?,a3-001,?)

The output of selecting from relation R using 1000001 and a3-001 with different indexing method

```
$ time ./select R
1000001,?,a3-001,? x
Query Stats:
# signatures read:   0
# sig pages read:    0
# tuples examined:   2000
# data pages read:   21
# false match pages: 20

real    0m0.004s
user    0m0.000s
sys     0m0.000s
```

```
$ time ./select R
1000001,?,a3-001,? t
Query Stats:
# signatures read:   2000
# sig pages read:    4
# tuples examined:   97
# data pages read:   1
# false match pages: 0

real    0m0.003s
user    0m0.000s
sys     0m0.000s
```

```
$ time ./select R
1000001,?,a3-001,? p
Query Stats:
# signatures read:   21
# sig pages read:    5
# tuples examined:   97
# data pages read:   1
# false match pages: 0

real    0m0.003s
user    0m0.000s
sys     0m0.000s
```

```
$ time ./select R
1000001,?,a3-001,? b
Query Stats:
# signatures read:   18
# sig pages read:    10
# tuples examined:   97
# data pages read:   1
# false match pages: 0

real    0m0.003s
user    0m0.000s
sys     0m0.000s
```

According to the output with different method, the running time of all kinds of method is the same.

Test case 2:

Description: This test case is based on relation R, with 8 attributes, 6000 expected tuples, 5000 actual inserted tuples and 1/2000 false match probably.

- First, create a relation R and insert 2000 tuples into it.

```
./create R 6000 8 2000
./gendata 5000 8 | ./insert R
```

- Select with open query (?,?,?,?,?,?,?,?)

The output of selecting everything from relation R with different indexing method

```
$ time ./select R
?,?,?,?,?,?,?,? x
Query Stats:
# signatures read:   0
# sig pages read:    0
# tuples examined:   5000
# data pages read:   87
# false match pages: 0

real        0m0.009s
user        0m0.004s
sys         0m0.000s
```

```
$ time ./select R
?,?,?,?,?,?,?,? t
Query Stats:
# signatures read:   5000
# sig pages read:    20
# tuples examined:   5000
# data pages read:   87
# false match pages: 0

real        0m0.022s
user        0m0.020s
sys         0m0.000s
```

```
$ time ./select R
?,?,?,?,?,?,?,? p
Query Stats:
# signatures read:   87
# sig pages read:    22
# tuples examined:   5000
# data pages read:   87
# false match pages: 0

real        0m0.022s
user        0m0.016s
sys         0m0.004s
```

```
$ time ./select R
?,?,?,?,?,?,?,? t
Query Stats:
# signatures read:   5000
# sig pages read:    20
# tuples examined:   5000
# data pages read:   87
# false match pages: 0

real        0m0.022s
user        0m0.016s
sys         0m0.004s
```

According to the output with different method, the running time of not using index is the fastest. Running time of using the others are the same.

- Select with querying one solution (1002000,?,?,?,?,?,?,?)

The output of selecting from relation R using id=1002000 with different indexing method

```
$ time ./select R
1002000,?,?,?,?,?,?,? x
Query Stats:
# signatures read:   0
# sig pages read:    0
# tuples examined:   5000
# data pages read:   87
# false match pages: 86

real        0m0.009s
user        0m0.004s
sys         0m0.000s
```

```
$ time ./select R
1002000,?,?,?,?,?,?,? t
Query Stats:
# signatures read:   5000
# sig pages read:    20
# tuples examined:   174
# data pages read:   3
# false match pages: 2

real        0m0.005s
user        0m0.000s
sys         0m0.004s
```

```
$ time ./select R
1002000,?,?,?,?,?,?,? p
Query Stats:
# signatures read:   87
# sig pages read:    22
# tuples examined:   58
# data pages read:   1
# false match pages: 0

real        0m0.004s
user        0m0.000s
sys         0m0.000s
```

```
$ time ./select R
1002000,?,?,?,?,?,?,? b
Query Stats:
# signatures read:   10
# sig pages read:    24
# tuples examined:   58
# data pages read:   1
# false match pages: 0

real        0m0.003s
user        0m0.000s
sys         0m0.000s
```

According to the output with different method, using bit-sliced index is the fastest, and then page signature, tuple signature and then not using index (in the order and it goes slower).

- Select with querying many solution (?,?,?,?,?,?,?,a8-101)

The output of selecting from relation R using a3-001 with different indexing method

```
$ time ./select R
?,?,?,?,?,?,?,a8-101 x
Query Stats:
# signatures read:   0
# sig pages read:    0
# tuples examined:   5000
# data pages read:   87
# false match pages: 79

real     0m0.009s
user     0m0.004s
sys      0m0.000s
```

```
$ time ./select R
?,?,?,?,?,?,?,a8-101 t
Query Stats:
# signatures read:   5000
# sig pages read:    20
# tuples examined:   580
# data pages read:   10
# false match pages: 2

real     0m0.007s
user     0m0.004s
sys      0m0.000s
```

```
$ time ./select R
?,?,?,?,?,?,?,a8-101 p
Query Stats:
# signatures read:   87
# sig pages read:    22
# tuples examined:   464
# data pages read:   8
# false match pages: 0

real     0m0.006s
user     0m0.004s
sys      0m0.000s
```

```
$ time ./select R
?,?,?,?,?,?,?,a8-101 b
Query Stats:
# signatures read:   10
# sig pages read:    24
# tuples examined:   464
# data pages read:   8
# false match pages: 0

real     0m0.003s
user     0m0.000s
sys      0m0.000s
```

According to the output with different method, the running time of using bit-sliced signature as index is a little bit faster than using page signature. Running time of the others are slower and basically the same.

- Select with querying multiple values (1000101,?,?,?,a5-001,?,?,?)

The output of selecting from relation R using 1000101 and a5-001 using different indexing method

```
$ time ./select R
1000101,?,?,?,a5-001,?,?,?
x
Query Stats:
# signatures read:   0
# sig pages read:    0
# tuples examined:   5000
# data pages read:   87
# false match pages: 87

real     0m0.009s
user     0m0.004s
sys      0m0.000s
```

```
$ time ./select R
1000101,?,?,?,a5-001,?,?,?
t
Query Stats:
# signatures read:   5000
# sig pages read:    20
# tuples examined:   0
# data pages read:   0
# false match pages: 0

real     0m0.005s
user     0m0.000s
sys      0m0.000s
```

```
$ time ./select R
1000101,?,?,?,a5-001,?,?,?
p
Query Stats:
# signatures read:   87
# sig pages read:    22
# tuples examined:   0
# data pages read:   0
# false match pages: 0

real     0m0.004s
user     0m0.000s
sys      0m0.000s
```

```
$ time ./select R
1000101,?,?,?,a5-001,?,?,?
b
Query Stats:
# signatures read:   20
# sig pages read:    24
# tuples examined:   0
# data pages read:   0
# false match pages: 0

real     0m0.002s
user     0m0.000s
sys      0m0.000s
```

According to the output with different method, using bit-sliced index is the fastest, and then page signature, tuple signature and then not using index (in the order and it goes slower).

Conclusion:

With the overall outputs, it is easy to find out that using bit-sliced signature is the fastest no matter how many data is in the relation. Using tuple signature and page signature usually take the same amount of time and they are the second better to query with large number of data, however, when testing using less number of data, querying without using any index is the second better.

I think the reason for fastest querying with bit-sliced is that it combines the good feature of page signature and bit string. In this case, querying could scan less number of signature and that saves the time.