SuRF: Practical Query Filtering with Fast Succinct Tries

Huanchen Zhang, Hyeontaek Lim, Viktor Leis, David G. Andersen, Michael Kaminsky, Kimberly Keeton, Andrew Pavlo

Presented by
Santhoshini Velusamy, Gabriele Oliaro, Eric Knorr, Chuqiao Yuan
Bloom filters

0 1 1 0 0

L0

1 1 0 0 1

L1

1 0 0 1 1

L2

Performs poorly for range queries!
Point Query

SELECT grade
FROM accounts
WHERE id = 641289

Range Query

SELECT grade
FROM accounts
WHERE 641289 <= id <= 641489

Existing partial solutions: B+ trees, prefix bloom filters.

A general-purpose filter that can handle both kinds of queries? SuRF!
SurF

- Speed
- Compact
- Accuracy
- Tunable false-positive rates
- Can handle both string and integer workloads!
- Order-preserving indexing

Order-preserving indexing
Fast Succinct Trie (FST)

Fast: much faster compared to previous succinct tries

Succinct: #nodes matches information theoretic bound.
10 bits per trie node!
LOUDS encoding
Level-Ordered Unary Degree Sequence encoding

Unary coding

Representing a natural number \( n \) with \( n \) ones followed by a zero

- 1: 10
- 5: 11110
- 10: 1111111110
Level-Ordered Unary Degree Sequence encoding

Unary coding

110 10 10 110 110 10 0 10 10 10 10 10 10 0 10 0 0
0 1 2 3 4 5 6 7 8 9 A B C D E F 10
LOUDS-DS encoding
LOUDS Dense-Sparse encoding
Rank and Select

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
10 & 10 & 10 & 110 & 110 & 10 & 0 & 10 \\
9 & 10 & 10 & 10 & 10 & 10 & 10 & 10
\end{array}
\]

Index 10

Index 17

\[
\text{Rank}_1(10) = 6 \quad \text{Select}_1(10) = 17
\]
Putting the Fast in FST

Rank & Select

Label Search

Prefetching

Label Search

128 bits, 16 values

==

128 bits, 16 values

32 bit value

32 bit value

32 bit value

LUT

B bits

B bits

B bits

Bit Vector

SIMD
The Trade-off Space

Accuracy

Memory

Computation
SuRF-Base:
Trimming Tries

Keys:
APOLLO
APPLE
BEATLE
BEE

Trimming Tries
SuRF-Hash:
Patch up Point-query Performance

<table>
<thead>
<tr>
<th>Keys:</th>
<th>Hash:</th>
</tr>
</thead>
<tbody>
<tr>
<td>APOLLO</td>
<td>0xE9EC34</td>
</tr>
<tr>
<td>APPLE</td>
<td>0xF0756D</td>
</tr>
<tr>
<td>BEATLE</td>
<td>0xB0CFCF</td>
</tr>
<tr>
<td>BEE</td>
<td>0x1D9DE9</td>
</tr>
</tbody>
</table>

Diagram:

```
    B
   / \  
  A   E
 /   /  
A   P   O
```

Hash:

```
A: 0xCF
B: 0x9D
E: 0x75
P: 0xEC
O: 0xD9
```
SuRF-Real:
Returning to Ranges

Keys:
APOLLO
APPLE
BEATLE
BEE

Returning to Ranges
FST evaluation - microbenchmark
FST evaluation - results

FST vs. Pointer-Based Indexes
FST evaluation - results

FST vs. Other Succinct Trees
FST evaluation - reason behind performance

FST Performance Breakdown

LOUDS-DS = LOUDS-Dense + LOUDS-Sparse
SuRF evaluation - false positive rate

Point Query

Range Query

Mixed Query

64-bit Integer

Email

No ordering information in Hash suffixes

Tunable to improve FPR in mixed query workloads
SuRF evaluation - performance

**Point Query**

- Throughput (Mops/s) vs. Bits per Key
- Comparison with Bloom Filter, SuRF-Hash, SuRF-Real, SuRF-Base

**Range Query**

- Throughput (Mops/s) vs. Bits per Key
- Comparison with SuRF-Base, SuRF-Real

**Mixed Query**

- Throughput (Mops/s) vs. Bits per Key
- Comparison with SuRF-Hash, SuRF-Real, SuRF-Mixed
System evaluation - benchmark

Time-series data  Key: 64-bit timestamp and 64-bit sensor ID
2K sensors       Value: 1KB
50K events

Total size: 100GB  Point query
                  Open-seek query
                  Closed-seek query
System evaluation - results

Figure 12: RocksDB point query and Open-Seek query evaluation under different filter configurations.

Closed-Seek – Throughput

Closed-Seek – I/O

Figure 13: RocksDB Closed-Seek query evaluation under different filter configurations and range sizes.
Possible next steps?
Thank you!