Improvement of Bloomfilters: A Rank and Selected Based Quotient Filter

Matthias Bungeroth

Chair for Database Systems - Tum

16th January 2018

Structure

Tasks

- Filters in general
- Bloom-filter
- Rank and Selected Based Quotient Filter
- Couting Rank and Selected Based Quotient Filter

Filters in general

Filters

- Can be configured with a false-positive-rate δ and n the element count to insert
- Implements method insert
- Implements method query that returns true or false

Filters in general

Counting-filters

- Implements method query that returns count

Bloom-Filter

A Bloom-filter is a couple (B, H).
With B a bit-vector and H a set of hash-functions.

Empty Bloom-Filter with H={ $h_1(x)$, $h_2(x)$ }

slot	0	1	2	3	4	5	6	7
B	0	0	0	0	0	0	0	0

Insert a and b with
$h_1(a) = 1, h_2(a) = 5$
$h_1(b) = 3, h_2(b) = 5$

slot	0	1	2	3	4	5	6	7
B	0	1	0	1	0	1	0	0

Counting-Bloom-Filter

A Bloom-filter is a couple (B , H).
With B a vector of counters and H a set of hash-functions.

slot	0	1	2	3	4	5	6	7
B	127	0	0	190	90	0	227	0

query(b) = 0
query(c) = 90

$h_1(b) = 3, h_2(b) = 5$
$h_1(c) = 3, h_2(c) = 4$

Rank and Selected Based Quotient Filter

- Spilts hash in h_0 (homeslot) and h_1 (remainder)
- Remainders are stored in homeslot if possible.

Rank and Selected Based Quotient Filter

- $\text{occupied}[x] = 1 \iff \exists_{y \in S} : h_0(y) = x$
- $\forall_{x,y \in S} : h_0(x) < h_0(y) \implies h1(x)$ is stored in an earlier slot than $h_1(y)$
- If $h_1(x)$ is stored in slot s, then $h_0(x) \leq s$ and there are no unused slots between slot $h_0(x)$ and slot s, inclusive.
- runends[b]=1 $\iff$ slot b contains the last remainder in a run.

S is a set of elements that have been inserted.

slot	0	1	2	3	4	5	6	7
occupied	0	0	0	0	0	0	0	0
runend	0	0	0	0	0	0	0	0
remainders	0	0	0	0	0	0	0	0

Rank and Selected Based Quotient Filter

Filters

slot	0	1	2	3	4	5	6	7
occupied	0	0	0	0	0	0	0	0
runend	0	0	0	0	0	0	0	0
remainders	0	0	0	0	0	0	0	0

$h_1(a) = 0$

Rank and Selected Based Quotient Filter

Insert-example

slot	0	1	2	3	4	5	6	7
occupied	1	0	0	0	0	0	0	0
runend	1	0	0	0	0	0	0	0
remainders	$h_1(a)$	0	0	0	0	0	0	0

$h_0(a) = 0$
$h_0(b) = 0$

Rank and Selected Based Quotient Filter

Insert-example

slot	0	1	2	3	4	5	6	7
occupied	1	0	0	0	0	0	0	0
runend	0	1	0	0	0	0	0	0
remainders	$h_1(a)$	$h_1(b)$	0	0	0	0	0	0

$h_0(b) = 0$
$h_0(c) = 0$
$h_0(d) = 0$

Rank and Selected Based Quotient Filter

Insert-example

slot	0	1	2	3	4	5	6	7
occupied	1	0	0	0	0	0	0	0
runend	0	0	0	1	0	0	0	0
remainders	$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	0	0	0	0

$h_0(c) = 0$
$h_0(d) = 0$
$h_0(e) = 1$

Rank and Selected Based Quotient Filter

Insert-example

slot	0	1	2	3	4	5	6	7
occupied	1	1	0	0	0	0	0	0
runend	0	0	0	1	1	0	0	0
remainders	$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	$h_1(e)$	0	0	0

$h_0(e) = 1$
$h_0(f) = 4$

Rank and Selected Based Quotient Filter

Rank and Select

$RANK(B, i) = \sum_{j=0}^{i} B[j]$ (Ammount of set bits in B upto postion i)

$SELECT(B, i) =$ (Index of the ith set bit in B)

Rank and Selected Based Quotient Filter

Insert-example

slot	0	1	2	3	4	5	6	7
occupied	1	1	0	0	0	0	0	0
runend	0	0	0	1	1	0	0	0
remainders	$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	$h_1(e)$	0	0	0

$h_0(f) = 4$

Rank and Selected Based Quotient Filter

Insert-example

slot	0	1	2	3	4	5	6	7
occupied	1	1	0	0	0	0	0	0
runend	0	0	0	1	1	0	0	0
remainders	$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	$h_1(e)$	0	0	0

$h_0(f) = 4$
$RANK(occupied, 4) = 2$
$SELECT(runend, 2) = 4$

Rank and Selected Based Quotient Filter

Insert-example

slot	0	1	2	3	4	5	6	7
occupied	1	1	0	0	1	0	0	0
runend	0	0	0	1	1	1	0	0
remainders	$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	$h_1(e)$	$h_1(f)$	0	0

$h_0(f) = 4$
$RANK(occupied, 4) = 2$
$SELECT(runend, 2) = 4$

Rank and Selected Based Quotient Filter

Insert-example

slot	0	1	2	3	4	5	6	7
occupied	1	1	0	0	1	0	0	0
runend	0	0	0	1	1	1	0	0
remainders	$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	$h_1(e)$	$h_1(f)$	0	0

$h_0(g) = 0$
$RANK(occupied, 0) = 1$
$SELECT(runend, 1) = 3$

Rank and Selected Based Quotient Filter

Insert-example

slot	0	1	2	3	4	5	6	7
occupied	1	1	0	0	1	0	0	0
runend	0	0	0	1	0	1	1	0
remainders	$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	0	$h_1(e)$	$h_1(f)$	0

$h_0(g) = 0$
$RANK(occupied, 0) = 1$
$SELECT(runend, 1) = 3$

Rank and Selected Based Quotient Filter

Insert-example

slot	0	1	2	3	4	5	6	7
occupied	1	1	0	0	1	0	0	0
runend	0	0	0	0	1	1	1	0
remainders	$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	$h_1(g)$	$h_1(e)$	$h_1(f)$	0

$h_0(g) = 0$

$RANK(occupied, 0) = 1$

$SELECT(runend, 1) = 3$

Rank and Selected Based Quotient Filter

Runend of slot

$$SELECT(runend, RANK(occupied, slot))$$

Returns corresponding runnend bit to a slot if occupied[slot]=1.

Rank and Selected Based Quotient Filter

Query

$$runend = SELECT(runend, RANK(occupied, slot))$$

slot	0	1	2	3	4	5	6	7
occupied	1	1	0	0	1	0	0	0
runend	0	0	0	0	1	1	1	0
remainders	$h_1(a)$	$h_1(b)$	$h_1(c)$	$h_1(d)$	$h_1(g)$	$h_1(e)$	$h_1(f)$	0

```
s = rankSelect(h0(x))
do{
   if remainders[s] = h1(x) then
      return true;
   s = s-1;
}while(s>h0(x) and !runend[s]);
return false;
```

Rank and Selected Based Quotient Filter

Improvement of runtime

- Linar runtime of Query and Insert cause by the Rank and Select operation
- Can be improved to 0(1) with offsets.

Rank and Selected Based Quotient Filter

Offsets

- $O_i = rankSelect(i) - i$
- Only defined if and only if occupied[i] = 1
- Only saved for every 64th slot
- To ensure every offset is defined runnend and occupied bits are inserted
- Save flag to check if element was inserted into a 64th slot

Rank and Selected Based Quotient Filter

Improvment of cache efficeny

- Currently all data is stored in different arrays
- Data can be reorganized into blocks

7	1	64	64	$r \cdot 64$
offset	used	occupieds	runends	remainders

Rank and Selected Based Quotient Filter

Counting

The Rank and Selected Based Quotient Filter counts unary.

slot	0	1	2	3	4	5	6	7
occupied	1	0	0	0	0	0	0	0
runend	0	0	0	0	0	0	1	0
remainders	$h_1(a)$	$h_1(a)$	$h_1(a)$	$h_1(a)$	$h_1(a)$	$h_1(a)$	$h_1(a)$	0

Counting Rank and Selected Based Quotient Filter

Counter encoding

- Encoded counters for elements can be added

slot	0	1	2	3	4	5	6	7
occupied	1	0	0	0	0	0	0	0
runend	0	0	1	0	0	0	0	0
remainders	$h_1(a)$	5	$h_1(a)$	0	0	0	0	0

Counting Rank and Selected Based Quotient Filter

Counter encoding

Count	Encoding	Rules
1	x	none
2	x, x	none
	For $x = 0$	
3	x, x, x	none
> 3	$x, c_{l-1}, \ldots, c_0, x, x$	$\forall_{c_i} \neq x$ $\forall_{i<l-1}\ c_i \neq x$
	For $x \neq 0$	
> 2	$x, c_{l-1}, \ldots, c_0, x$	$x > 0$ $c_{l-1} < x$ $\forall_{i<l-1}\ c_i \neq x$ $\forall_{c_i} \neq x$

Counting Rank and Selected Based Quotient Filter

Counter encoding

For $x \neq 0$ and count $C \geq 3$:
$C - 3$ as $c_{l-1}, \ldots, c_0$ in base $2^r - 2$ where symbols are
$1, 2, \ldots, x-1, x+1, \ldots, 2^r - 1$ and attach a zero to front if $c_l > x$.

For $x = 0$ and count $C \geq 4$:
$C - 4$ as $c_{l-1}, \ldots, c_0$ in base $2^r - 1$ where symbols are $1, 2, \ldots, 2^r - 1$.

Evaluation

Rank and Selected Based Quotient Filter variants

- Runtime
- Space consumption

Evaluation

Runtime

- Random inserts
- Queries on inserted elements
- Random queries

Evaluation

Rank and Selected Based Quotient Filter variants

Configuration	Operations	RSQF no	RSQF nb	RSQF
$\delta = 0.001$	Random insert	20s	$<$5ms	$<$2ms
$n = 10000$	Query on inserted elements	20s	$<$5ms	$<$2ms
	Random query(100% load)	0.1s	$<$1ms	$<$0.5ms
$\delta = 0.0001$	Random insert	/	1.4s	3.7s
$n = 10000000$	Query on inserted elements	/	1.8s	5.3s
	Random query(100% load)	/	0.7s	0.6s
$\delta = 0.001$	Random insert	/	43s	15s
$n = 100000000$	Query on inserted elements	/	52s	17s
	Random query(100% load)	/	8.2s	7.3s

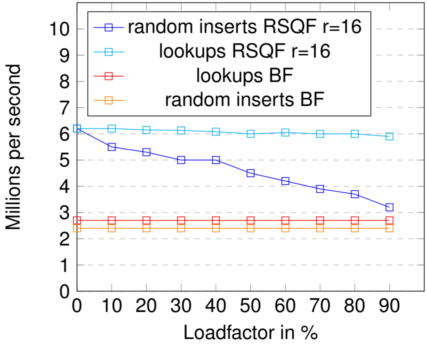
Evaluation

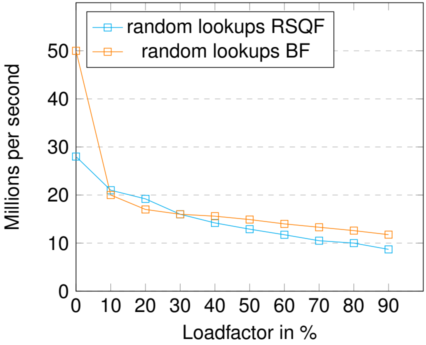
Rank and Selected Based Quotient Filter compared to Bloomfilter

Configuration	**Operations**(in million per second)	**BF**	**RSQF**
			(r=4)
$\delta = 0.01$	Random insert	2.9	6.0
$n = 10000000$	Query on inserted elements	3.2	7.6
	Random query(100% load)	12.7	12.0
			(r=8)
$\delta = 0.00001$	Random insert	1.6	8.8
$n = 10000000$	Query on inserted elements	1.8	6.6
	Random query(100% load)	12.26	25.7
			(r=16)
$\delta = 0.000001$	Random insert	1.1	4.7
$n = 100000000$	Query on inserted elements	1.3	5.0
	Random query(100% load)	10.0	10.4

Evaluation

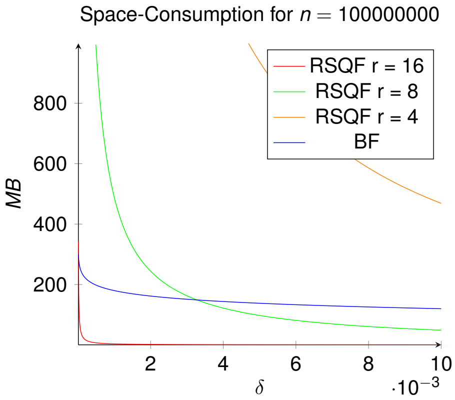
Rank and Selected Based Quotient Filter variants





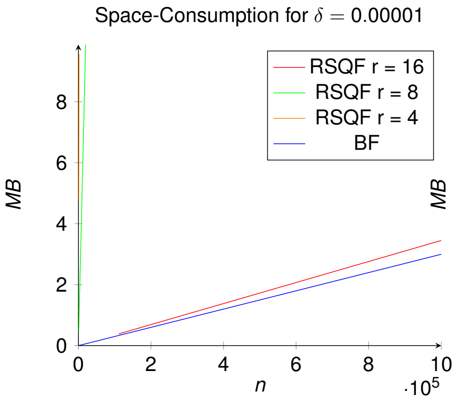
Evaluation

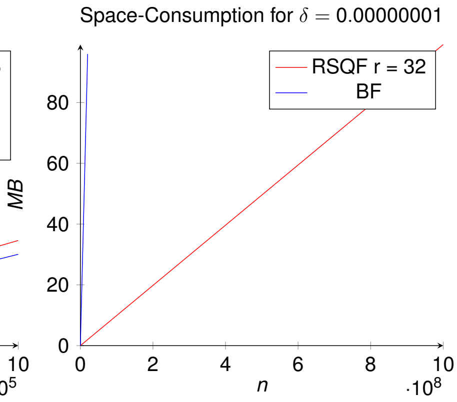
Space-analysis



Evaluation

Space-analysis





Evaluation

Counting-filter

Operations(in million per second)	**CBF**	**CQF**(r=8)
Random insert	7.9	13.5
Random lookup	7.7	9.6

Table showing average runtime of 1000 000 000 operations each for a CQF/ CBF configured with $\delta = 0.0001$,$n = 2000$

Thanks for your attention.

Any Questions?

Implementation....