I’ve got a database in Brooklyn to sell you.

Timon Karnezos
Neustar
SF PostgreSQL User’s Group
2014-09-23

a crude overview of sketching

history of hyperloglog

postgresql-hll

resources for further study

Background

a <u>crude</u> definition of
probabilistic & streaming
algorithms

streaming setting:
small (sublinear) memory
one pass over data
constant update time

(silly) streaming algorithms:
max, min, mean

probabilistic algorithm:
inject reproducible randomness
“smooth out” average case

sketching = streaming & probabilistic:
approximate answer
error bound holds with some prob.
[nice to have: additivity]

POST /nasal/drip
@moonpolysoft

Follow

smugly points out that something is no panacea

Reply Retweet Favorite ••• More

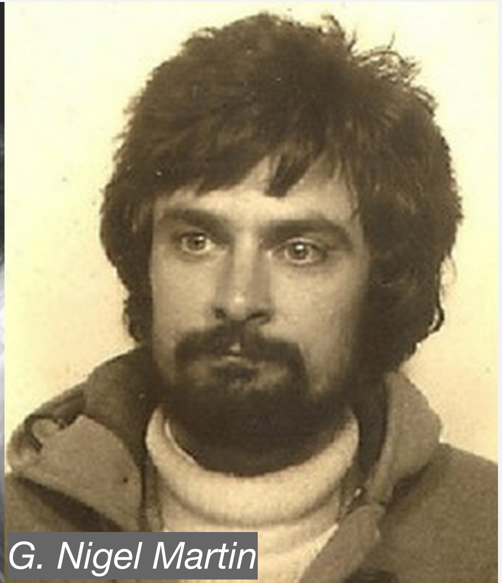
RETWEETS 3 FAVORITES 9

7:41 AM - 28 Aug 2014

History

“Probabilistic Counting”

1983



→ RDBMS research in 70s
→ automatic query planning
→ need selectivity estimates
→ need cardinality estimates

“Probabilistic Counting”

1983

count to N

with $\log_2(N)$ bits

"Probabilistic Counting"

1983

count to N

with $\log_2(N)$ bits

Usually 2^{32}

Hint: 32

"Probabilistic Counting"

1983

intuition:

if i flip a coin a bunch of times, and tell you I saw <u>10 heads in a row</u> at some point, how many times did i toss that coin?

“Probabilistic Counting”

1983

Assume $N = 2^8$
for this example.

“Probabilistic Counting”

1983

Assume $h(v)$
is a “good” hash
function.

"Probabilistic Counting"

1983

Assume $h(v)$ is a "good" hash function.

Map from domain D to $\{0,1\}^L$ for some large enough L (usually 32) whose output is uniformly random.

“Probabilistic Counting”

1983

0	0	0	0	0	0	0	0
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*

“Probabilistic Counting”

1983

$h(v_0) = 10000000$

→ hash values to $\{0,1\}^L$

0_0	0_1	0_2	0_3	0_4	0_5	0_6	0_7

"Probabilistic Counting"

1983

$h(v_0) = 10000000 = \text{run of length } 0$

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

→ hash values to $\{0,1\}^L$

→ track runs of lead zeroes

“Probabilistic Counting”

1983

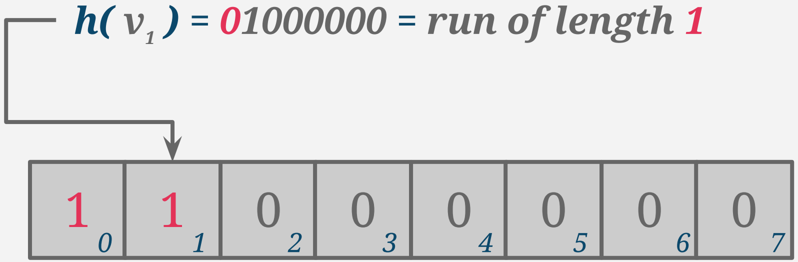
$h(v_0) = 10000000 = \text{run of length } 0$

1	0	0	0	0	0	0	0
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*

→ hash values to $\{0,1\}^L$
→ track runs of lead zeroes
→ mark run length in bitmap

“Probabilistic Counting”

1983

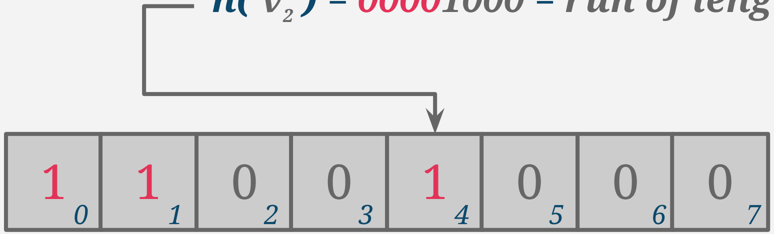


- → hash values to $\{0,1\}^L$
- → track runs of lead zeroes
- → mark run length in bitmap

“Probabilistic Counting”

1983

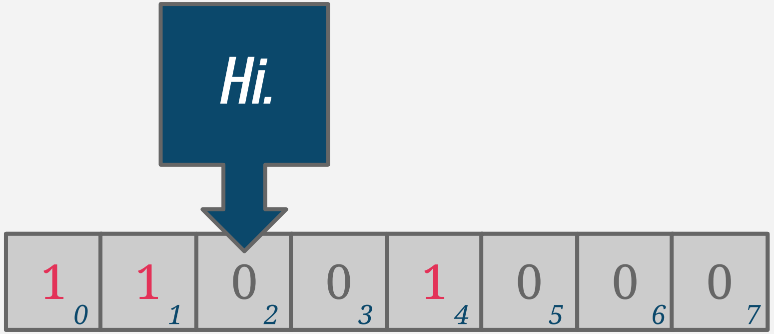
$h(v_2) = 00001000$ = *run of length 4*



→ hash values to $\{0,1\}^L$
→ track runs of lead zeroes
→ mark run length in bitmap

“Probabilistic Counting”

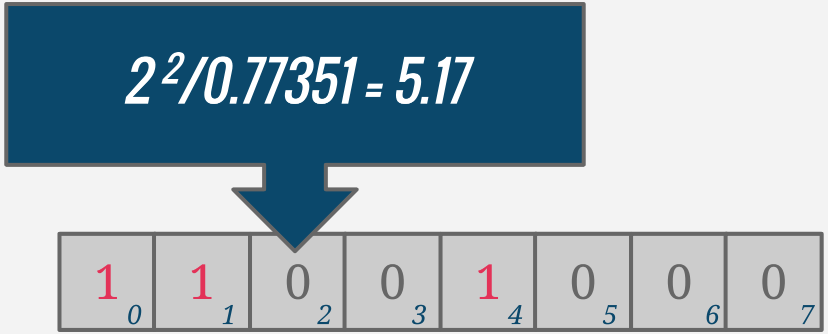
1983



→ hash values to $\{0,1\}^L$
→ track runs of lead zeroes
→ mark run length in bitmap
→ find index of left-most zero

“Probabilistic Counting”

1983



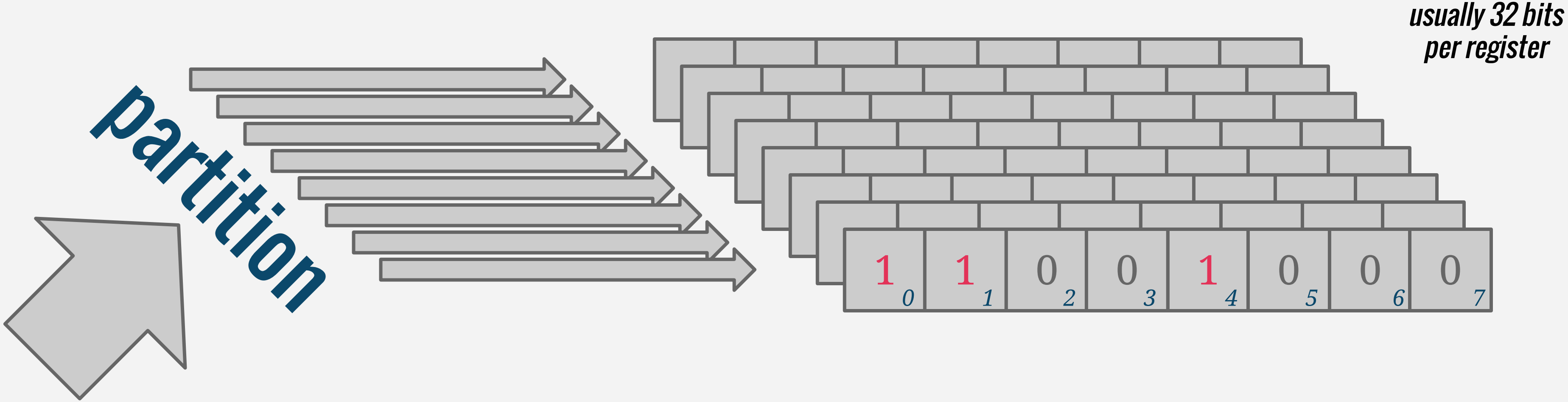
- → hash values to $\{0,1\}^L$
- → track runs of lead zeroes
- → mark run length in bitmap
- → find index of left-most zero
- → cardinality: $2^i/\phi$

“Probabilistic Counting”

1983

so that an estimate based on (1) will typically be one binary order of magnitude off the exact result, a fact that calls for more elaborate algorithms to be developed in Section 3.

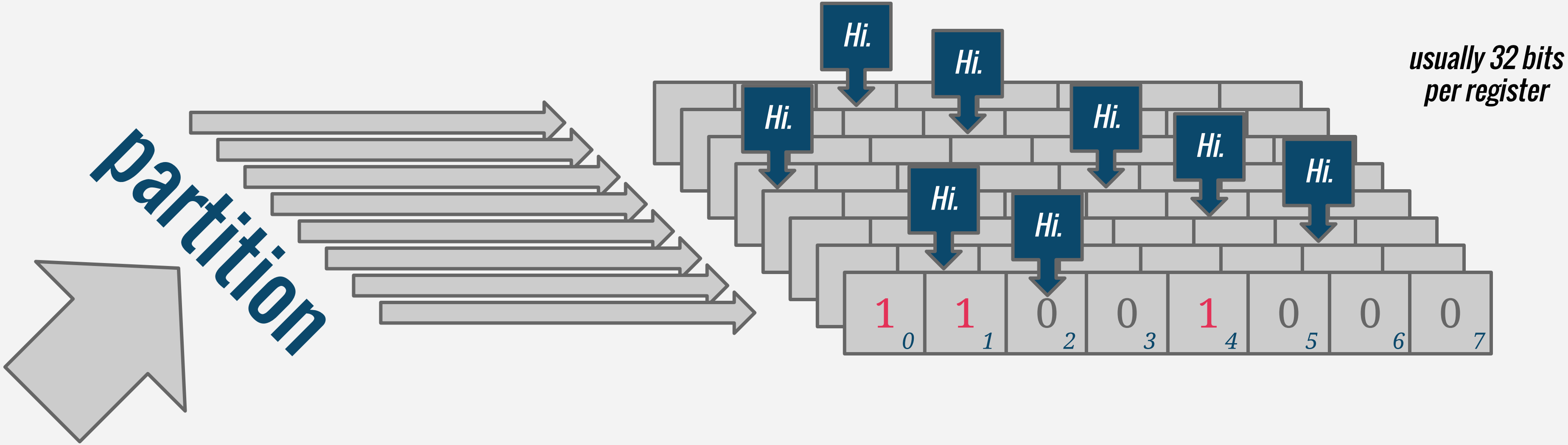
“Probabilistic Counting”



“... with Stochastic Averaging”

1985

“Probabilistic Counting”



“... with Stochastic Averaging”

1985

"Probabilistic Counting"

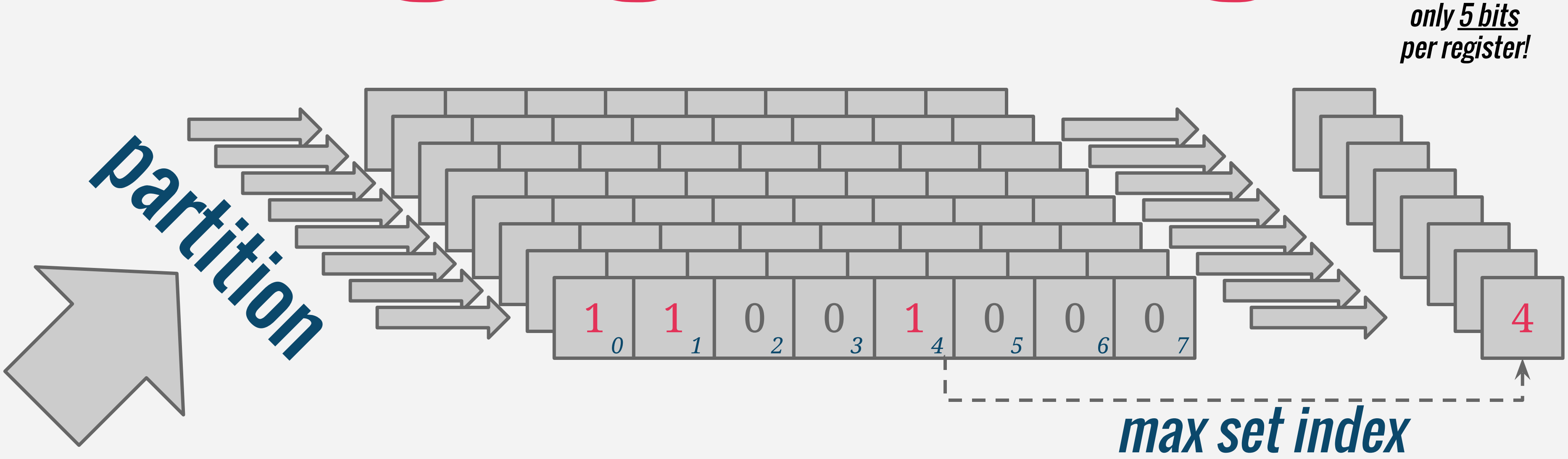
error bounded by:

0.78/sqrt(substream count)

"... with Stochastic Averaging"

1985

“LogLog Counting”



“... of Large Cardinalities”

2003

"LogLog Counting"

error bounded by:

1.3/sqrt(substream count)

"... of Large Cardinalities"

2003

"LogLog Counting"

Jesus!

z is a positive real. The function $ub(z)$ is equal to $e^z(1 + z2^{-l})$.

Proof of proposition 2 Maple gives us a nice expression for the integral of f.

$$\int_{2^l}^{\infty} f(x)dx = 2^{l/m} \sum_{k \geq 1} \frac{1}{k - 1/m} \frac{1}{k!} \left((-n/2^l)^k - (-2n/2^l)^k\right).$$

"... of Large Cardinalities"

2003

“HyperLogLog” 2007

$$\boxed{\alpha_m m 2^{\left(\frac{\sum_i M_i}{m}\right)}}$$

compute $Z := \left(\sum_{j=1}^{m} 2^{-M[j]}\right)^{-1}$; {*the “indicator” function*}

return $E := \alpha_m m^2 Z$ with α_m as given by Equation (3).

→ same data structure as LogLog
→ better mean of register values
(arithmetic to harmonic mean)
→ tighter error bounds

Enough theory!

postgresql-hll

- →code
- →design
- →examples
- →data brag
- →lessons learned

postgresql-hll

- → 2500 lines of C
- → 500 lines of SQL
- → 1000 lines of comments
- → Austin Appleby's C++ Murmur3
- → 55MB test vectors

postgresql-hll

- →marshal to/from bytea
- →bit slicing to update registers
- →formula for cardinality
- →union(hll_1, hll_2)

postgresql-hll

compact, combinable, approximate unique counts of users

postgresql-hll

compact, combinable, approximate

Hierarchical storage format

→empty token (3 bytes)

→explicit list of hashes (8 bytes x configurable)

→hashmap of register index to register value (...)

→full array of registers representation (5 x 2^{m-3} bytes)

postgresql-hll

compact, combinable, approximate

<u>Additivity allows:</u>

→union (“seen A or seen B”)

  →union preserves relative error

→set difference* (“seen A but not B”)

→intersections* (“seen A and B”)

*use sparingly! non-linear error propagation! (bit.ly/hllinter)

postgresql-hll

compact, combinable, approximate

<u>Relative error:</u>

→2^{14} x 5-bit registers = 81920 bits = 10kB

→1% relative error

→e.g. 1B uniques x 1% = ±10M absolute count error

examples

daily_uniques		
Column	**Type**	**Meaning**
`report_date`	`date`	day of counts
`impressions`	`bigint`	number of page views
`users`	`hll`	set of unique cookie ids

examples

```
SELECT
  report_date,
  impressions,
  #users
FROM daily_uniques
WHERE report_date BETWEEN
    '...' AND '...'
```

examples

```
SELECT report_date,
       SUM(impressions)         OVER last7 AS imps_cumu,
       #hll_union_agg(users) OVER last7 AS users_cumu,
       imps_cumu/users_cumu                AS avg_frequency
FROM daily_uniques
WINDOW last7 AS
   (ORDER BY report_date ASC ROWS 6 PRECEDING)
WHERE report_date BETWEEN '...' AND '...'
ORDER BY report_date ASC
```

examples

For more examples, see:

bit.ly/pghll

bragging rights

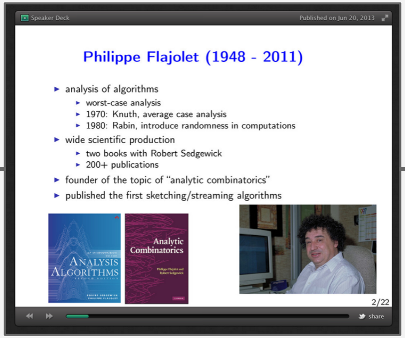
- → PG 9.3
- → MMs new hll instances/day
- → hll_union_agg 1M rows ~20s
- → Java interop via java-hll
- → Been doing this for 4+ years

lessons learned

→Pick a good non-cryptographic hash
→Don’t mess with inputs
→Rigorously unit and fuzz test interop
→Leave crumbtrails to the paper in source

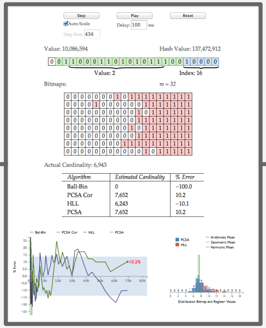
I am extremely grateful
to the following persons
for their contributions to
both **this talk** and to our
open source efforts.

History: bit.ly/lumbroso



Jérémie Lumbroso

Sketching Vis: bit.ly/sketchotd



Rob Grzywinski
Matt Curcio
Ben Linsay

Streaming book: bit.ly/muthubook

S. Muthukrishnan

Streaming ppt: bit.ly/andonippt

Alex Andoni

HLL physics: bit.ly/chenderson

Chris Henderson

Hashing: bit.ly/pesyna

Colin Pesyna

@alberts
@blinsay
@jdmaturen
@metdos
@ozgune
@yerenkow

github.com/aggregateknowledge/{postgresql,java,js}-hll

Papers

- →MJRTY ('81)
- →Probabilistic Counting ('83)
- →Probabilistic Counting with Stochastic Averaging ('85)
- →LogLog (and SuperLogLog) ('03)
- →CountMin Sketch ('05)
- →HyperLogLog ('07)
- →K Min Values ('07)

Other Materials

- →Notes/Lectures from DIKU Summer School on Hashing ('14)
  - →Mikkel and Michael's talks are fantastic.
  - →In fact, just go read everything Michael's ever written on sketching
    - → {{Invertible, Compressed, Counting} Bloom, Cuckoo} {filters, tables}

I WILL PERSONALLY BRIBE YOU TO MAKE POSTGRESQL-HLL GO FASTER.

SSE/SIMD,
toast magic,
marshalling magic,
WHATEVER MAGIC YOU GOT.

THANK YOU!

Timon Karnezos
@timonk
research.neustar.biz