I’ve got a database in Brooklyn to sell you.
a crude overview of sketching

history of hyperloglog

postgresql-hll

resources for further study
Background
a crude 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]
*smugly points out that something is no panacea*
“Probabilistic Counting”

Philippe Flajolet  G. Nigel Martin

→ RDBMS research in 70s
→ automatic query planning
→ need selectivity estimates
→ need cardinality estimates
count to $N$ with $\log_2(N)$ bits
"Probabilistic Counting"

Count to \( N \) with \( \log_2(N) \) bits

Hint: 32

Usually \( 2^{32} \)
intuition:
if I flip a coin a bunch of times, and tell you I saw 10 heads in a row at some point, how many times did I toss that coin?
Assume $N = 2^8$ for this example.
Assume $h(v)$ is a “good” hash function.
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”
"Probabilistic Counting"

\( h( v_0 ) = 10000000 \)

\[ \rightarrow \text{hash values to } \{0,1\}^L \]
“Probabilistic Counting”

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

→ hash values to $\{0,1\}^L$

→ track runs of lead zeroes
"Probabilistic Counting"

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

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

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

\[ h(v_1) = 01000000 = \text{run of length 1} \]
"Probabilistic Counting"

1983

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

\[ h(v_2) = 00001000 = \text{run of length 4} \]
“Probabilistic Counting”

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

Hi.
"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: $2^i/\phi$

$2^2/0.77351 = 5.17$
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"

partition

usually 32 bits per register

1985
“Probabilistic Counting”

“... with Stochastic Averaging”
“Probabilistic Counting”

error bounded by:

0.78/sqrt(substream count)

“... with Stochastic Averaging”
"LogLog Counting"

"... of Large Cardinalities"

partition

max set index

only 5 bits per register!

2003
“LogLog Counting”

error bounded by:

1.3/sqrt(substream count)

“... of Large Cardinalities”
\[ z \text{ is a positive real. The function } \vartheta_b(z) \text{ is equal to } e^z(1 + z2^{-l}). \]

**Proof of proposition 2** Maple gives us a nice expression for the integral of \( f \).

\[
\int_{2^l}^{\infty} f(x) \, dx = 2^{l/m} \sum_{k \geq 1} \frac{1}{k-1/m} \frac{1}{k!} \left( (-n/2^l)^k - (-2n/2^l)^k \right).
\]
“HyperLogLog”

2007

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

\[ \alpha_m m^2 \left( \frac{\sum_i M_i}{m} \right) \]

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

return \( E := \alpha_m m^2 Z \) with \( \alpha_m \) as given by Equation (3).
Enough theory!
postgresql-hll

→ code
→ design
→ examples
→ data brag
→ lessons learned
→ 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(\text{hll}_1, \text{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

Additivity allows:
→ 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)
compact, combinable, approximate

Relative error:
→ $2^{14} \times 5$-bit registers = 81920 bits = 10kB
→ 1% relative error
→ e.g. 1B uniques x 1% = ±10M absolute count error
### daily_uniques

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>report_date</td>
<td>date</td>
<td>day of counts</td>
</tr>
<tr>
<td>impressions</td>
<td>bigint</td>
<td>number of page views</td>
</tr>
<tr>
<td>users</td>
<td>hll</td>
<td>set of unique cookie ids</td>
</tr>
</tbody>
</table>
SELECT
    report_date,
    impressions,
    #users
FROM daily_uniques
WHERE report_date BETWEEN
    '...' AND '....'
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
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.
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)
→ 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