"Probabilistic" Data Structures vs. PostgreSQL

(and similar stuff)

FOSDEM PgDay - January 31, 2020

Tomas Vondra
tomas.vondra@2ndquadrant.com
tv@fuzzy.cz / @fuzzycz

HyperLogLog and t-digest

Probabilistic data structures ... use hash functions to randomize and compactly represent a set of items.

These algorithms use much less memory and have constant query time … and can be easily parallelized.

https://dzone.com/articles/introduction-probabilistic-0 https://en.wikipedia.org/wiki/Category:Probabilistic_data_structures

Bloom Filter	(set membership)
HyperLogLog	(count distinct)
Count-Min Sketch	(frequency table)
MinHash	(set similarity)
…	
… random trees, heaps, ...	

https://en.wikipedia.org/wiki/Category:Probabilistic_data_structures

● Bloom Filter	(set membership)
● HyperLogLog	(count distinct)
● Count-Min Sketch	(frequency table)
● MinHash	(set similarity)
● …	
● … random trees, heaps, ...	

access_log

```
CREATE TABLE access_log (
    ...
    req_date         TIMESTAMPTZ,
    user_id          INTEGER,
    response_time    DOUBLE PRECISION,
    ...
);

CREATE TABLE access_log (req_date timestamptz, user_id int,
response_time double precision);

INSERT INTO access_log SELECT i, 1000000 * random(), 1000 *
random() from generate_series('2019-01-01'::timestamptz,
'2020-02-01'::timestamptz, '1 second'::interval) s(i);
```

```
SELECT COUNT(DISTINCT user_id)
FROM access_log
```

COUNT(DISTINCT user_id)

- has to deduplicate data
- needs a lot of memory / disk space
- … so it's slow
- difficult to precalculate
- difficult to compute incrementally
- difficult to parallelize

HyperLogLog

HyperLogLog

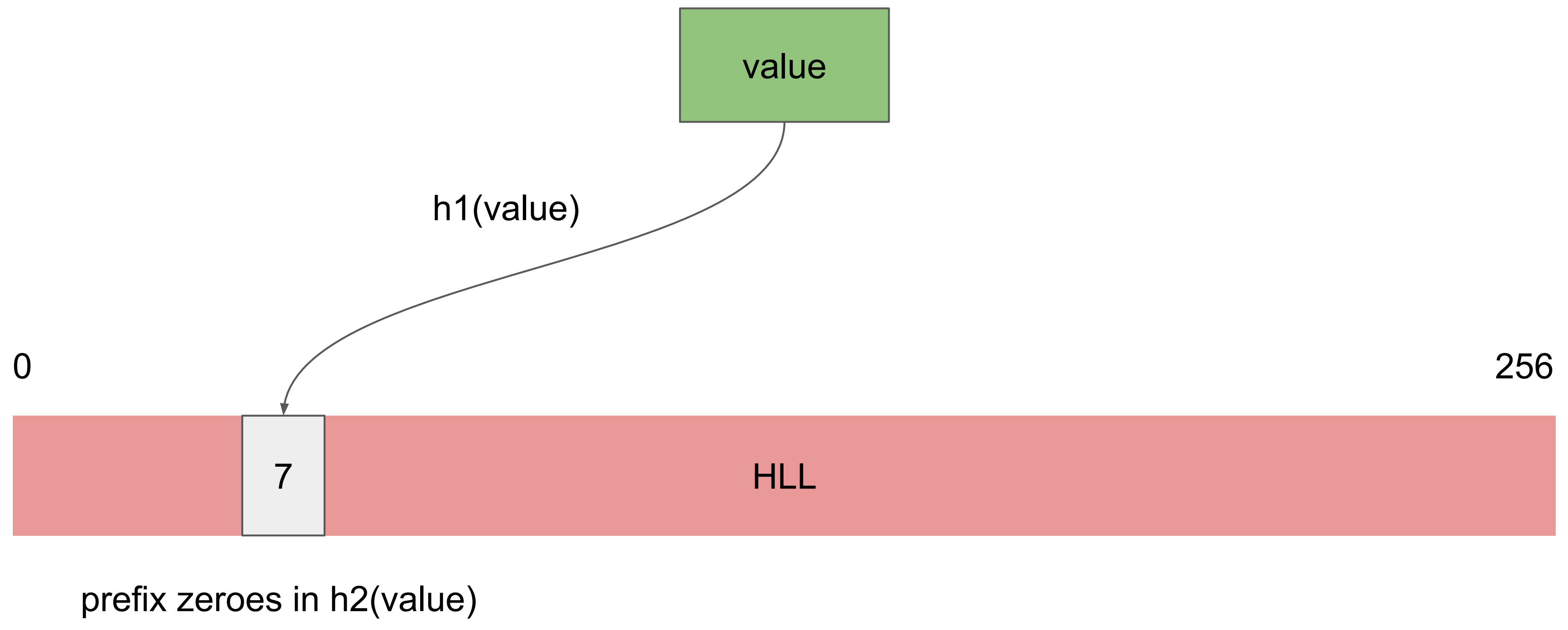
- when it's enough to have (accurate) estimate

```
SELECT COUNT(DISTINCT user_id) FROM access_log;
```

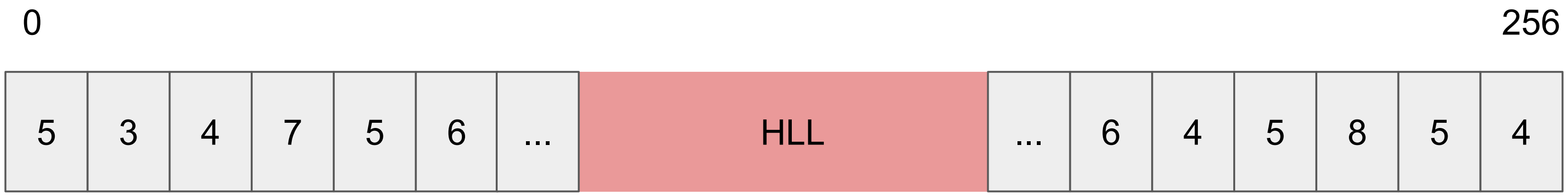
- we'll observe number of zeroes at the beginning of the hash value
  - `1xxxxxxxx` => 1/2
  - `01xxxxxxx` => 1/4
  - ...
  - `0000001xx` => 1/128

- Maximum number of zeroes we've seen is 6. What's the cardinality?

HyperLogLog



HyperLogLog



harmonic mean + correction

https://github.com/citusdata/postgresql-hll

Alternative to COUNT(DISTINCT user_id)

```
-- install the extension
CREATE EXTENSION hll;

-- generate HLL counter from user_id values
SELECT hll_add_agg(hll_hash_integer(user_id))
  FROM access_log;

-- estimate the cardinality of user_id values
SELECT #hll_add_agg(hll_hash_integer(user_id))
  FROM access_log;
```

Rollup (pre-calculation)

```
-- create a rollup table
CREATE TABLE access_log_daily (req_day date,
req_users hll);


-- pre-calculate daily summaries
INSERT INTO access_log_daily
SELECT
  date_trunc('day', req_date),
  hll_add_agg(hll_hash_integer(user_id))
FROM access_log
GROUP BY 1;
```

Rollup (pre-calculation)

```
-- use the rollup to summarize range
SELECT #hll_union_agg(req_users)
  FROM access_log_daily
 WHERE req_day BETWEEN '2019-10-01' AND
'2019-10-08';
```

HyperLogLog

- 2007 (evolution from ~1990)
- just an estimate, not an exact cardinality
  - but you can compute the maximum error
- trade-off between size and accuracy
  - size grows very slowly (with increasing accuracy / number of values)
  - 6kB more than enough for 1B values with 1% accuracy (1.5kB - 2% etc.)
- supports
  - precalculation (rollup)
  - incremental updates
  - ...

t-digest

percentile_cont / percentile_disc

```
SELECT
   percentile_cont(0.95)
      WITHIN GROUP (ORDER BY response_time)
FROM access_log
```

percentile_cont / percentile_disc

```
SELECT
   percentile_cont(ARRAY[0.95, 0.99])
      WITHIN GROUP (ORDER BY response_time)
FROM access_log
```

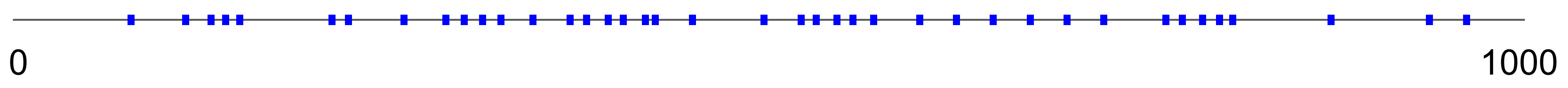
percentile_cont / percentile_disc

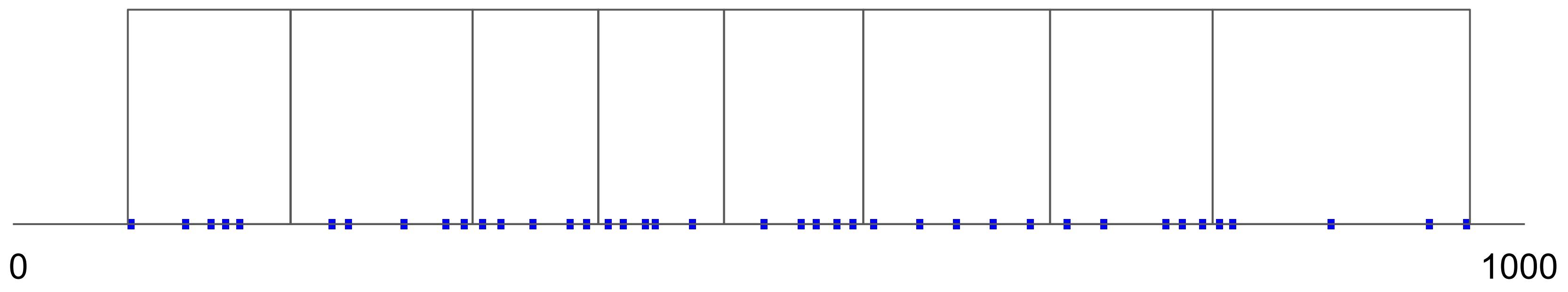
- accurate results
- has to store and sort all the data
- difficult to parallelize
- can't be precalculated

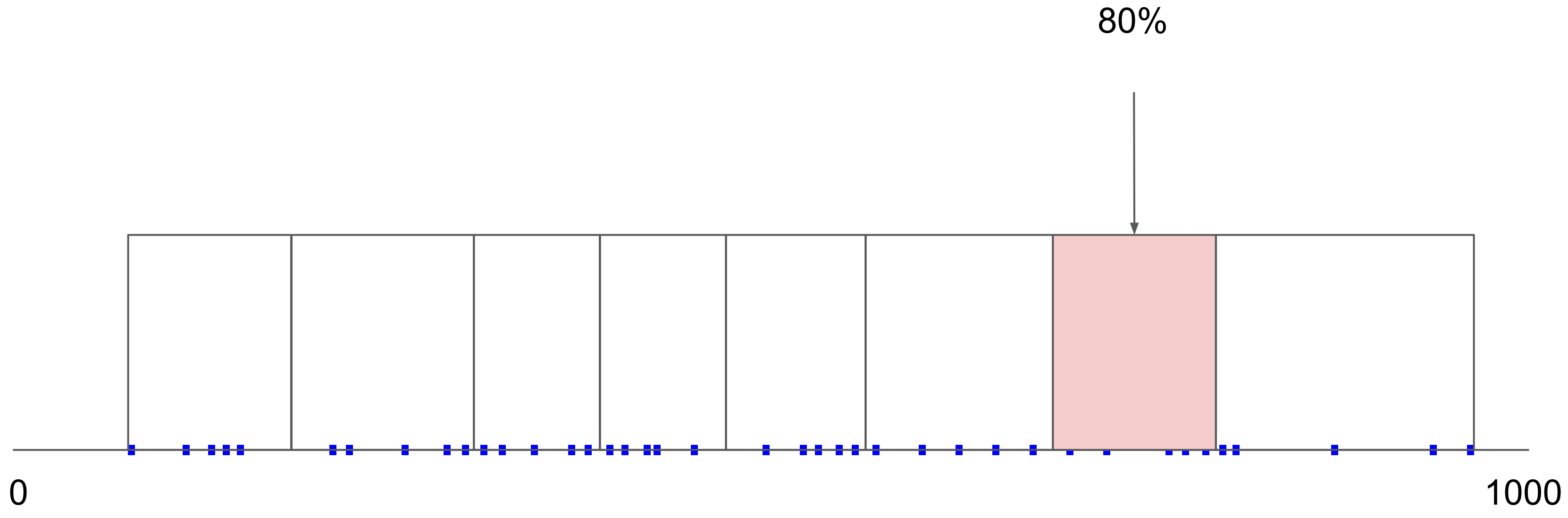
:-(

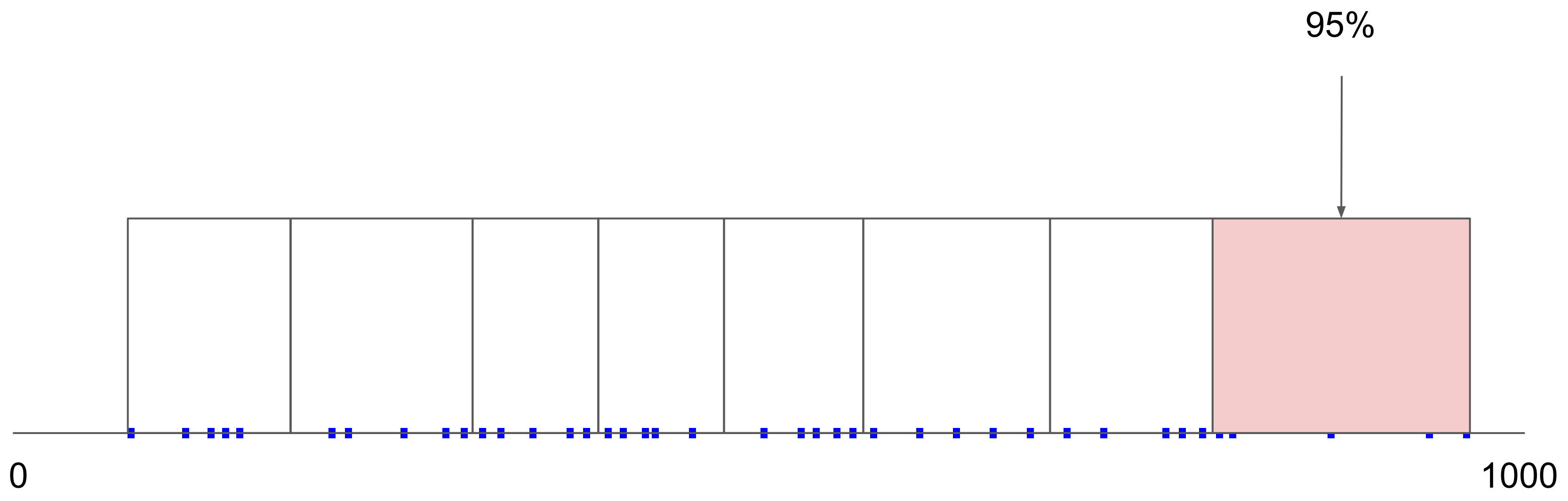
t-digest

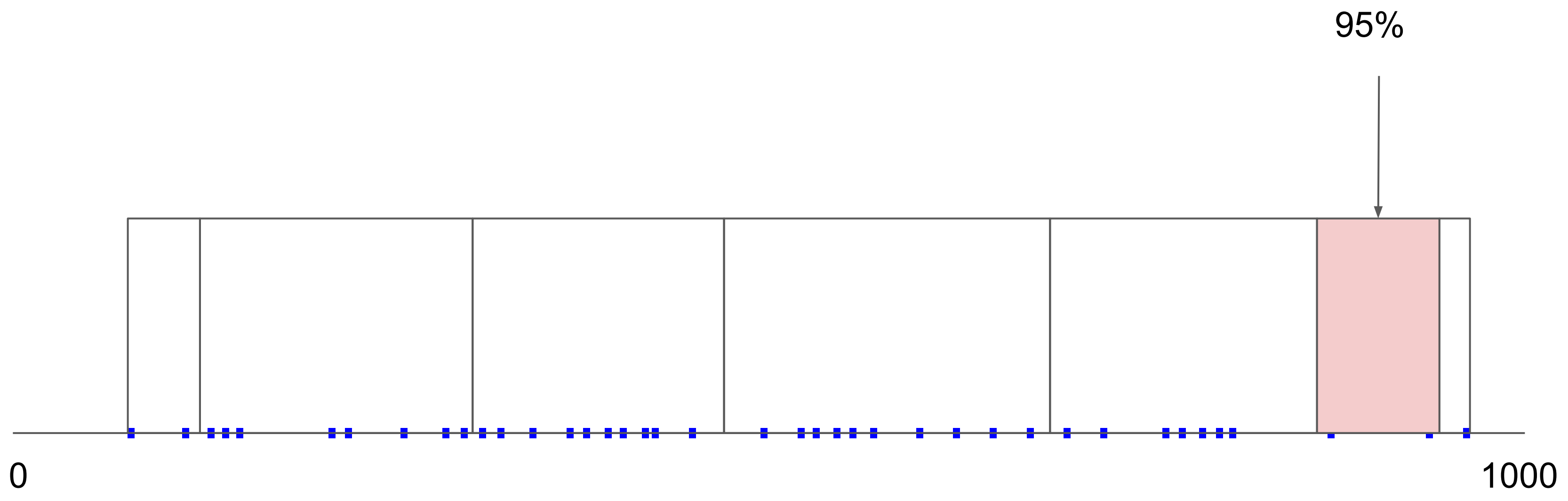
- published in 2013 by Ted Dunning
- approximation of CDF (cumulative distribution function)
- essentially a histogram
  - represented by centroids, i.e. each bin is represented by [mean, count]
  - requires data types with ordering and mean
- intended for stream processing
  - but hey, each aggregate is processing a stream of data
- higher accuracy on the tails (close to 0.0 and 1.0)

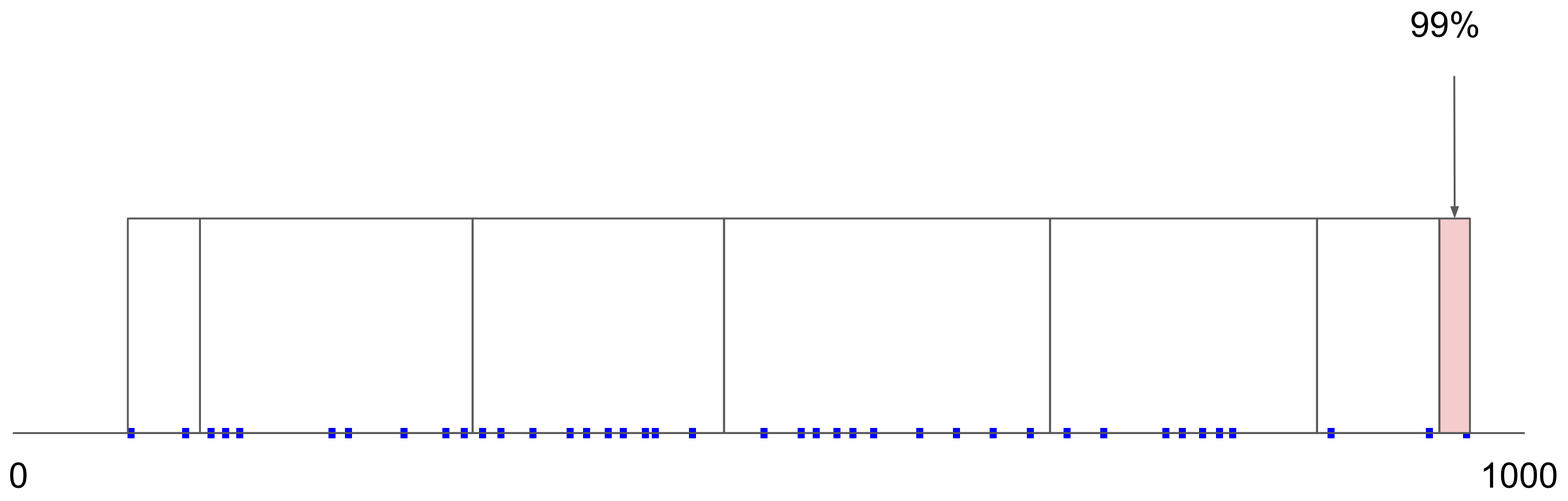

0
1000

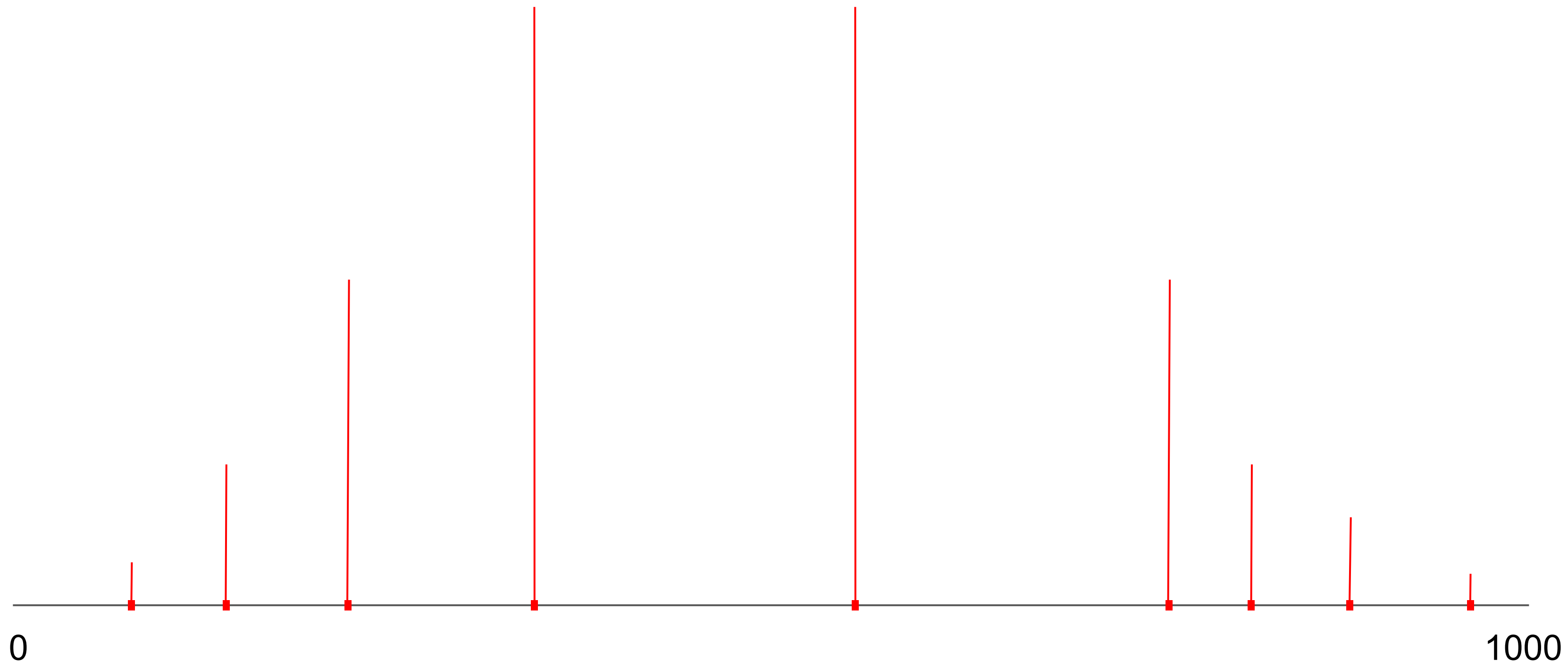

0
1000

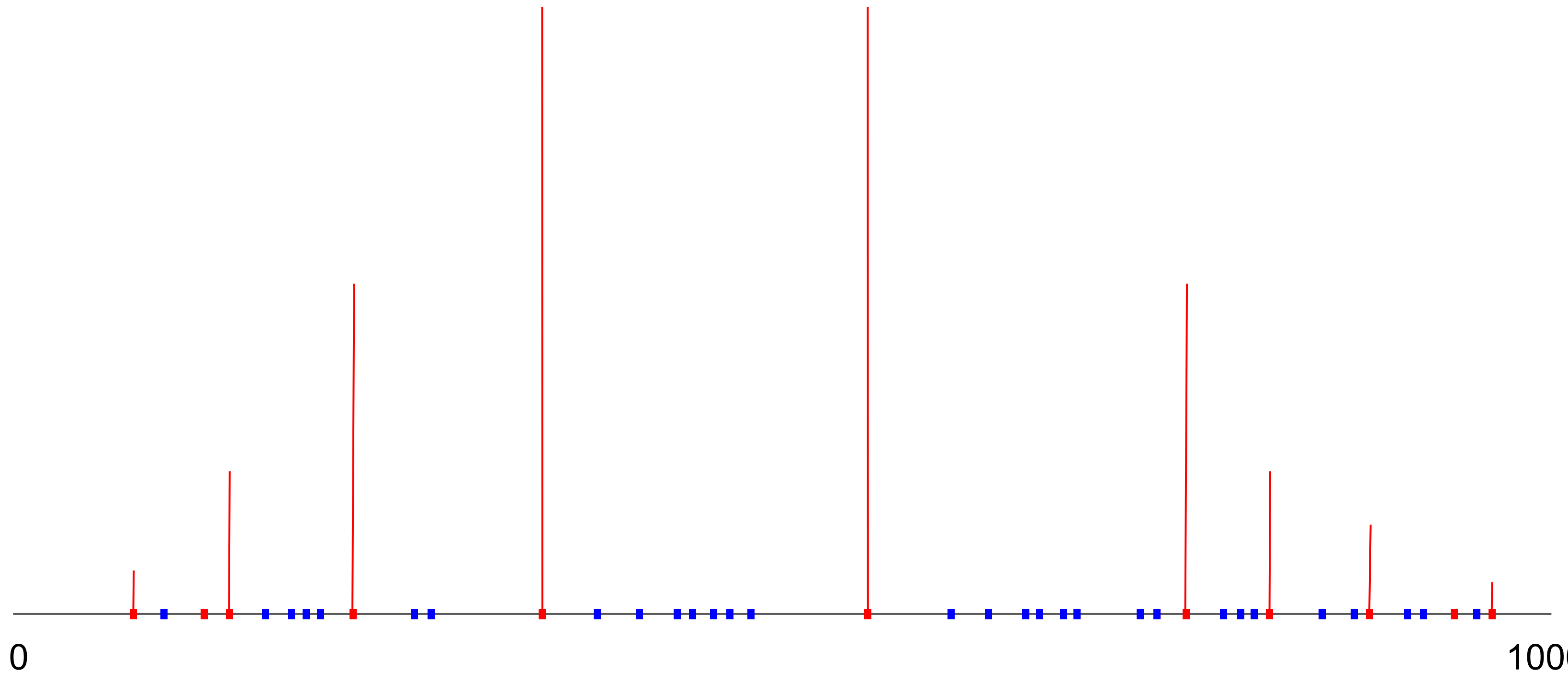

80%
0
1000


95%
0
1000


95%
0
1000


99%
0
1000


0
1000


0
1000

https://github.com/tvondra/tdigest

Trivial example

```
SELECT
   percentile_cont(0.95)
      WITHIN GROUP (ORDER BY response_time)
FROM access_log



SELECT
   tdigest_percentile(response_time, 100, 0.95)
FROM access_log
```

Precalculation

```
CREATE TABLE precalc_digests (
   req_day         date,
   req_durations    tdigest
);



INSERT INTO precalc_digests
SELECT
   date_trunc('day', req_date),
   tdigest(response_time, 100)
FROM access_log GROUP BY 1;
```

t-digest

- modus operandi similar to HyperLogLog
  - approximation by simpler / smaller data structure
  - incremental updates
  - possibility to precalculate + rollup
- result depends on order of input values
  - affects parallel queries
- no formal accuracy limits
  - better accuracy on tails
  - worse accuracty close to 0.5 (median)

?