estimating percentiles

(and calculating them faster)

Tomáš Vondra <tomas@vondra.me>, Microsoft
@tomasv@fosstodon.org

Nordic PGDay 2026, March 24, Helsinki

Agenda

- not a new invention / feature
  - research papers, stream databases, ...
- not strictly database (or Postgres) topic
  - motivation: stream processing
  - but it's interesting and fun!
- calculating exact percentiles
  - ... and what are the challenges
- sketches
  - general idea
  - sketches for percentiles: t-digest / ddsketch

Why this talk?

Microsoft

- distraction from "normal" work
  - Postgres features
- sketches are a really interesting concept
  - approximation vs. accurate results
  - ... and not just for percentiles!
- show Postgres extensibility
  - it's not hard* to do this inside Postgres
  - pretty much "simple" aggregate functions
  - you get the rest as a bonus

percentiles

Microsoft

- 50% (median), 95%, 99%, ...
  - often used in SLAs / monitoring
  - simple average / min / max does not describe the data
- exact calculation is (very) expensive
  - requires two passes (and expensive sort)
  - can't precalculate on subsets of data (how would you merge?)
  - difficult to change resolution (1s, 1m, 1h, ...) or apply filters
  - have to keep all the data (doesn't work for stream processing)
- if you do this once a day, that's fine
  - but if you do it on the main dashboard, it's not great

percentile_cont / percentile_disc

https://www.postgresql.org/docs/current/functions-aggregate.html#FUNCTIONS-ORDEREDSET-TABLE

```
CREATE TABLE request_timings (req_method INT, req_time DOUBLE PRECISION);

-- table with 10M rows, first column has 1000 distinct values
INSERT INTO request_timings
SELECT mod(i,1000), 100 * random()
  FROM generate_series(1, 10_000_000) s(i);

VACUUM ANALYZE request_timings;

-- calculate median and 95th percentile for each "method"
SELECT req_method,
   percentile_cont(ARRAY[0.5, 0.95]) WITHIN GROUP (ORDER BY req_time)
FROM request_timings GROUP BY req_method;
```

percentile_cont / percentile_disc

https://www.postgresql.org/docs/current/functions-aggregate.html#FUNCTIONS-ORDEREDSET-TABLE

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 GroupAggregate  (cost=1658507.15..1733518.60 rows=1000 width=36)
   Group Key: req_method
   ->  Sort  (cost=1658507.15..1683506.80 rows=9999860 width=12)
         Sort Key: req_method
         ->  Seq Scan on request_timings  (cost=0.00..154053.60 rows=9999860 width=12)
(5 rows)

Time: 3693.776 ms
```

percentile_cont / percentile_disc

https://www.postgresql.org/docs/current/functions-aggregate.html#FUNCTIONS-ORDEREDSET-TABLE

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=431168.44..1678509.89 rows=1000 width=36)
   Group Key: req_method
   ->  Gather Merge  (cost=431168.44..1628498.09 rows=9999860 width=12)
         Workers Planned: 4
         ->  Sort  (cost=430168.39..436418.30 rows=2499965 width=12)
               Sort Key: req_method
               ->  Parallel Seq Scan on request_timings  (cost=0.00..79054.65 rows=2499965 width=12)
(7 rows)

Time: 2869.319 ms (serial: 3693.776 ms)
```

percentile_cont / percentile_disc

https://www.postgresql.org/docs/current/functions-aggregate.html#FUNCTIONS-ORDEREDSET-TABLE

```
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=431168.44..1678509.89 rows=1000 width=36)
   Group Key: req_method
   ->  Gather Merge  (cost=431168.44..1628498.09 rows=9999860 width=12)
         Workers Planned: 4
         ->  Sort  (cost=430168.39..436418.30 rows=2499965 width=12)
               Sort Key: req_method
               ->  Parallel Seq Scan on request_timings  (cost=0.00..79054.65 rows=2499965 width=12)
(7 rows)

Time: 2869.319 ms (serial: 3693.776 ms)
```

THIS IS NOT THE RIGHT SORT!

Microsoft

sketches / digests

What are sketches?

Microsoft

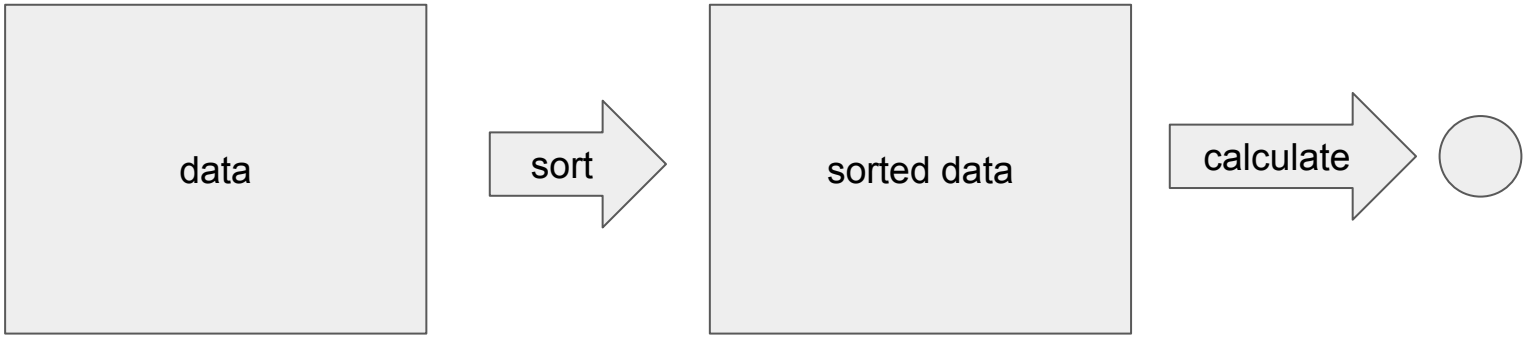
- simplified representation of the data
  - estimates, not exact answers
  - construction depends on purpose
  - some based on probability observations (hll counters)
- incremental building
  - enough to see data points once (stream processing)
- advanced features
  - "merging" of sketches (parallelism, filtering)
- percentile sketches
  - trivial sketch: MCV
  - state of the art: t-digest, ddsketch

sketch examples

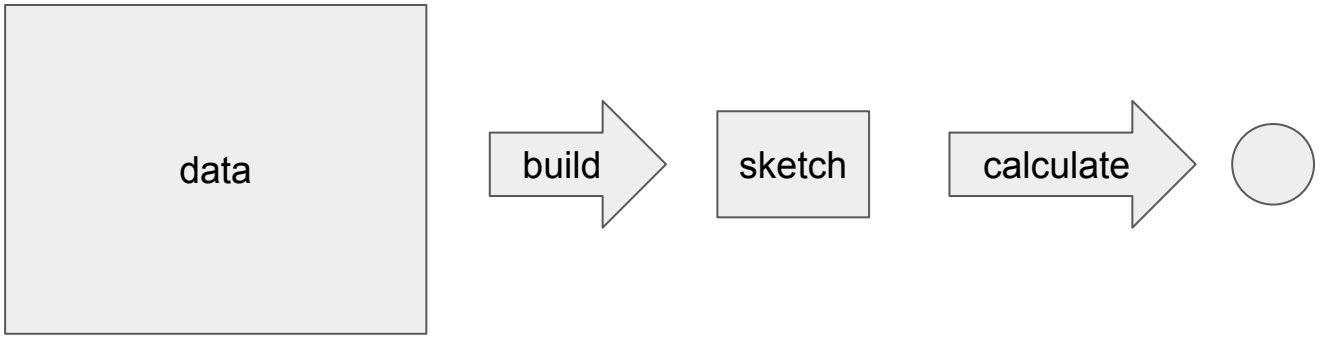
Microsoft

- hyperloglog [https://github.com/citusdata/postgresql-hll]
  - approximates COUNT(DISTINCT x)
  - similar capabilities, probabilistic foundation
- OpenHistogram (Hartmann, Schlossnagle)
  - https://openhistogram.io/
- HdrHistogram
  - https://github.com/HdrHistogram/HdrHistogram
- count-min sketch
  - ...
- omnisketch
  - ...

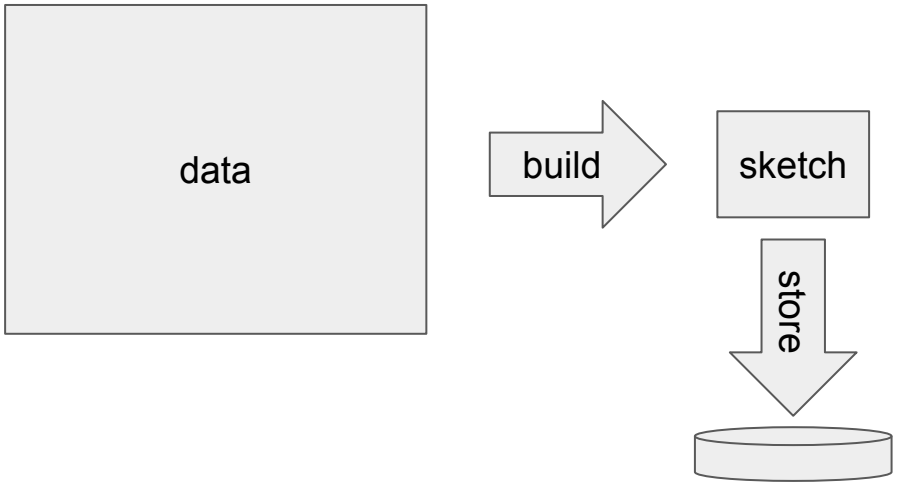
The "traditional" calculation



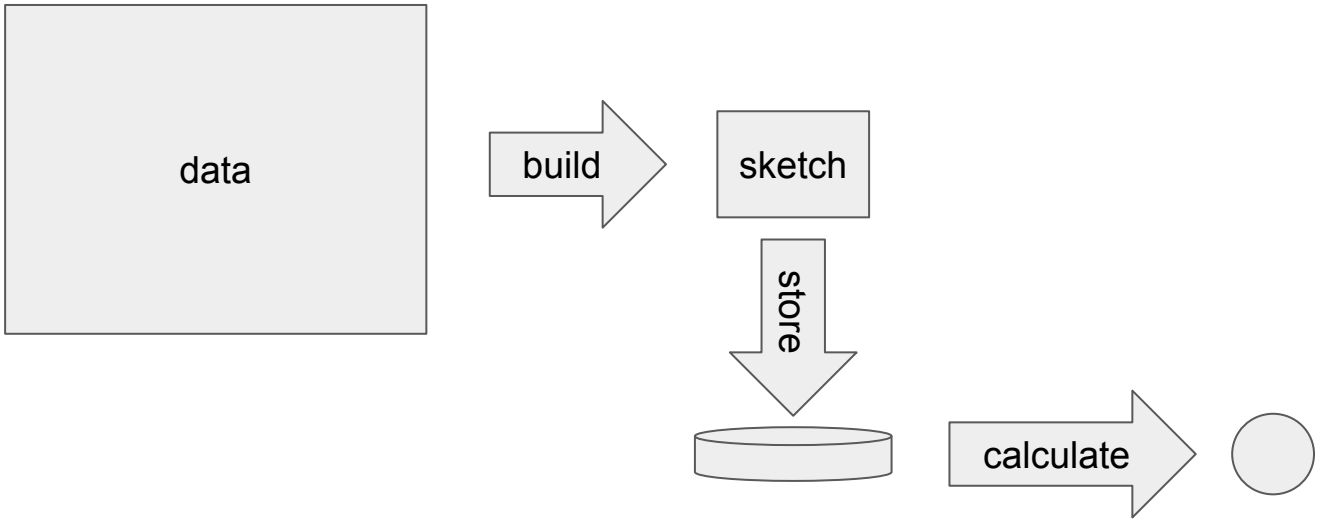
Calculation with sketches



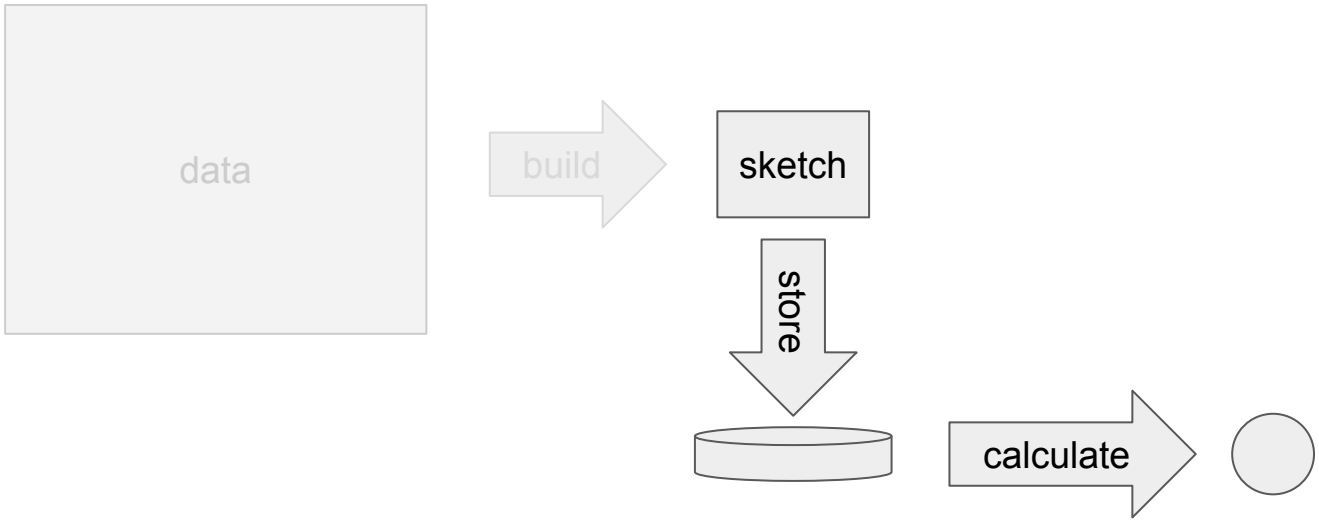
Calculation with sketches



Calculation with sketches

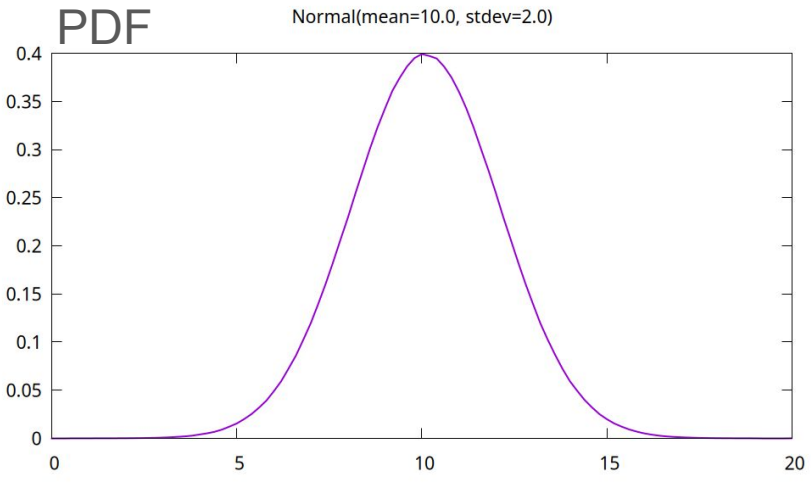


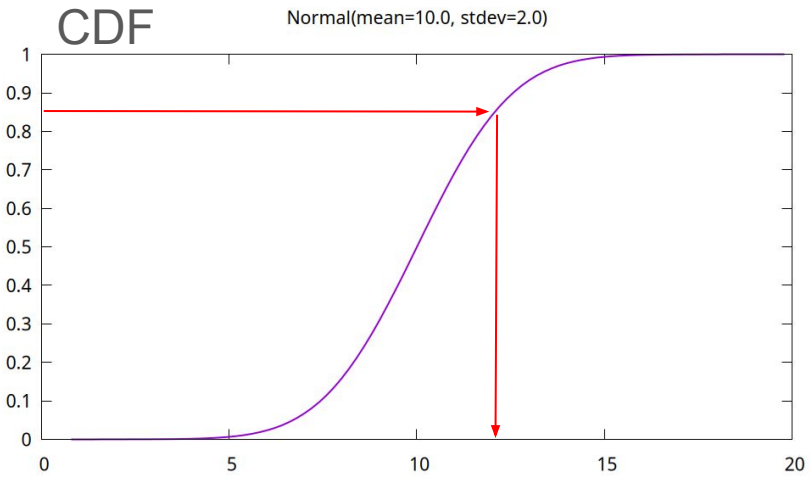
Calculation with sketches



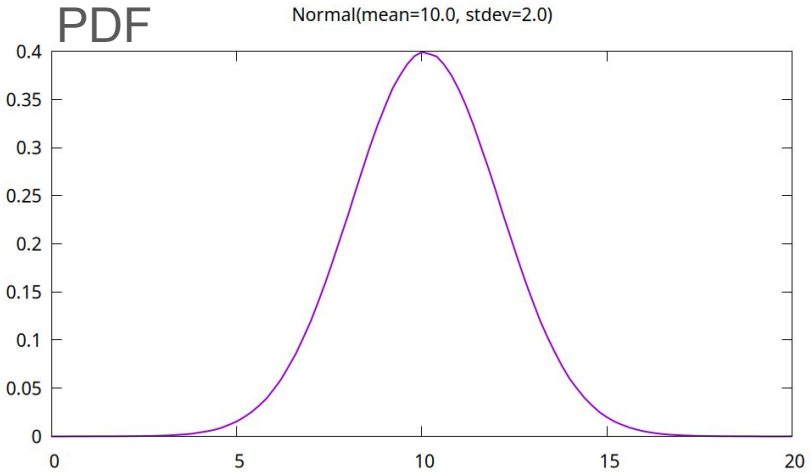
Microsoft

percentiles vs. histograms

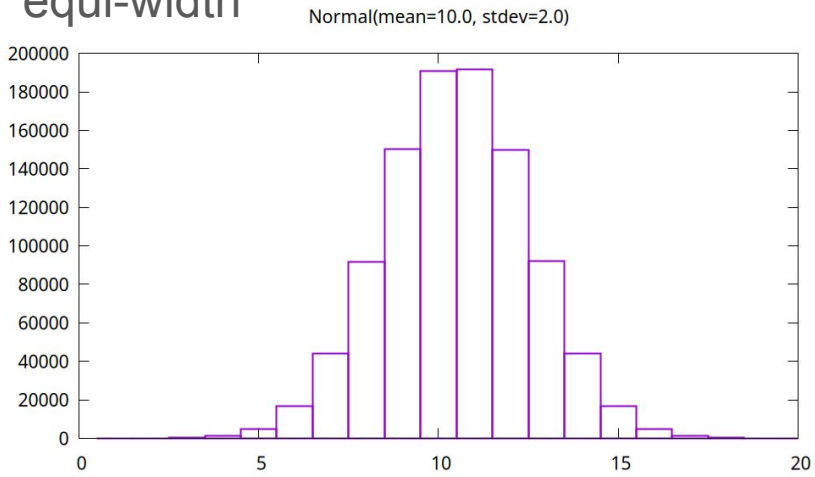

PDF
Normal(mean=10.0, stdev=2.0)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20


CDF
Normal(mean=10.0, stdev=2.0)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20

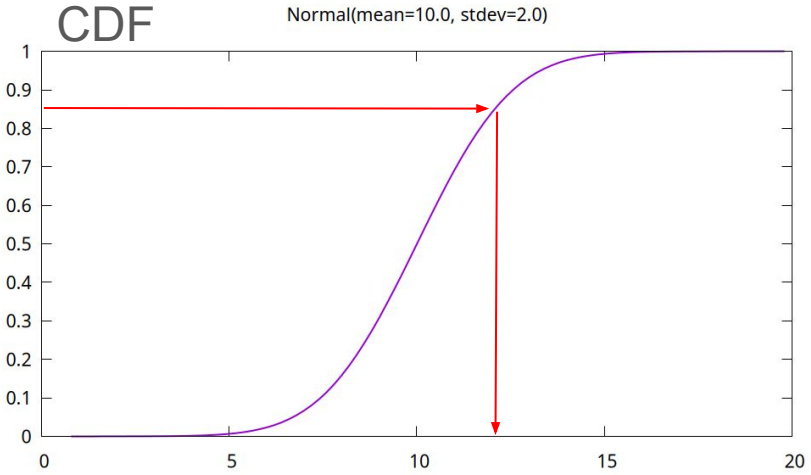
PDF



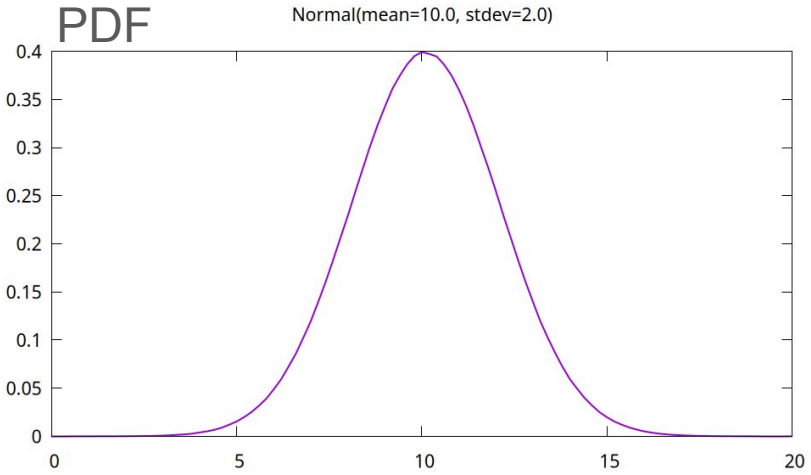
equi-width



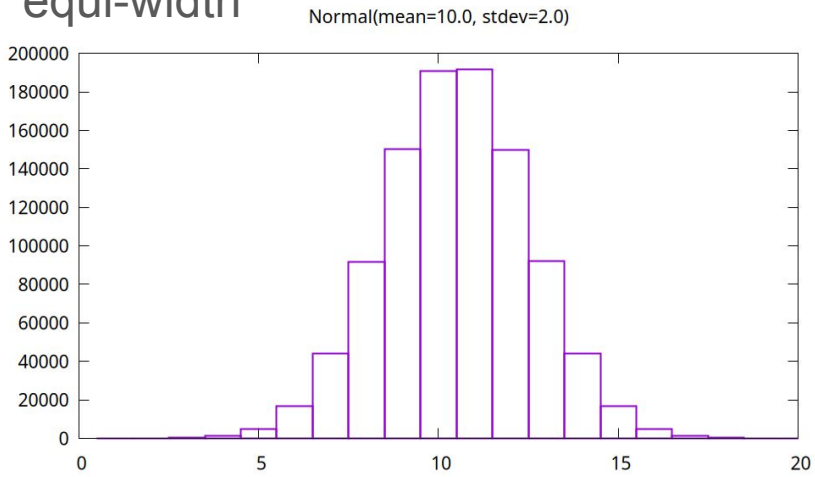
CDF



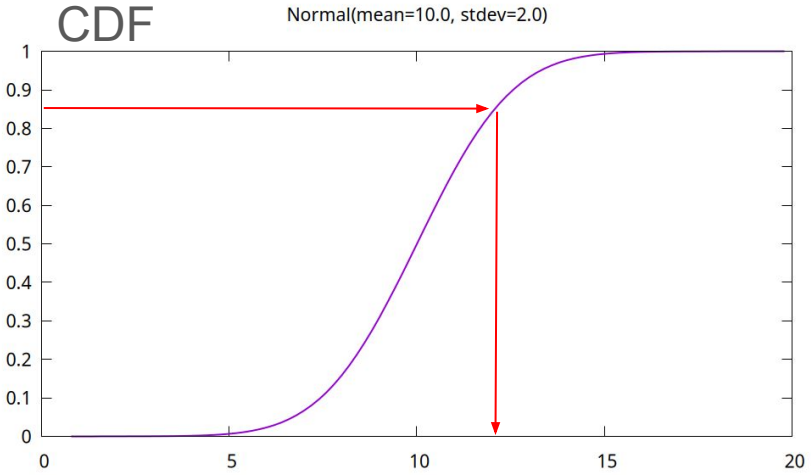
PDF



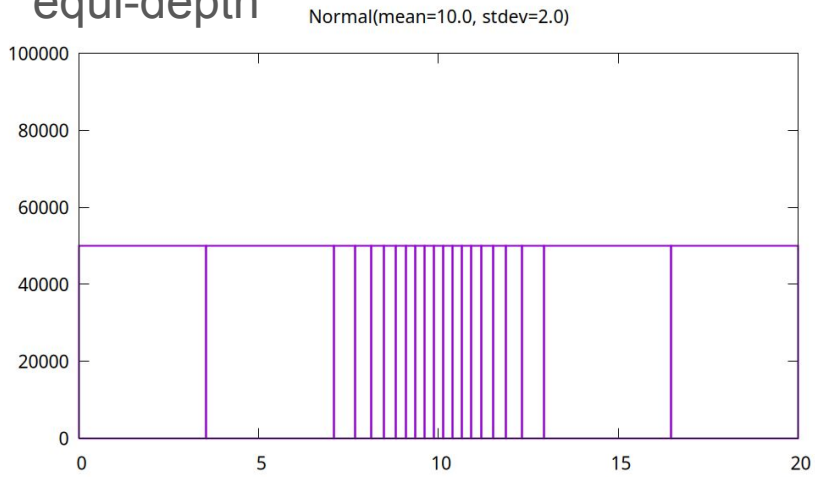
equi-width



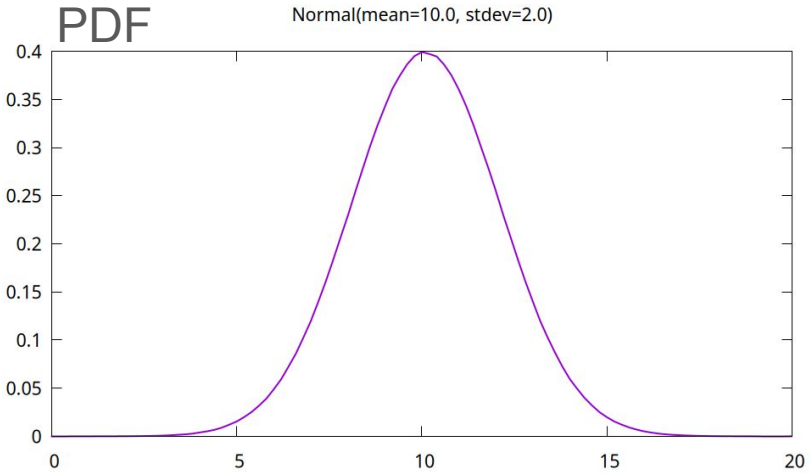
CDF



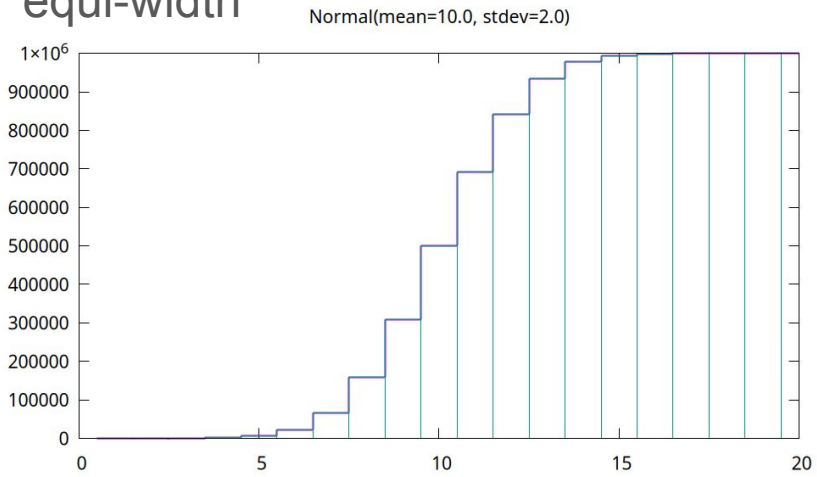
equi-depth



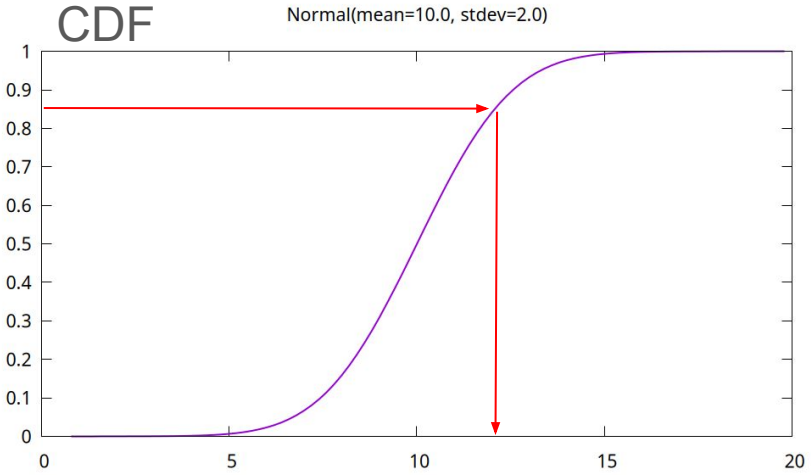
PDF



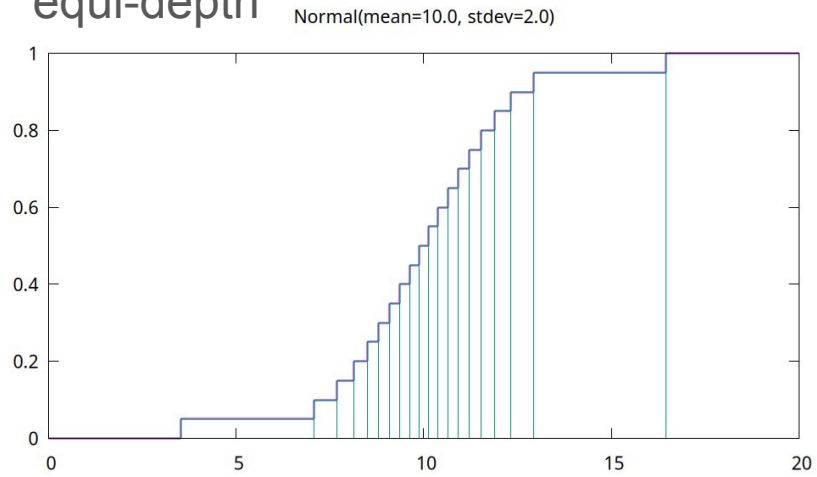
equi-width

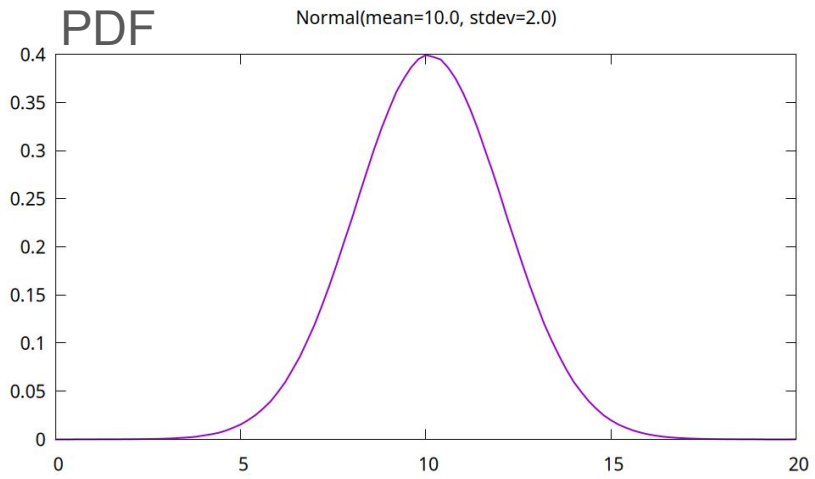


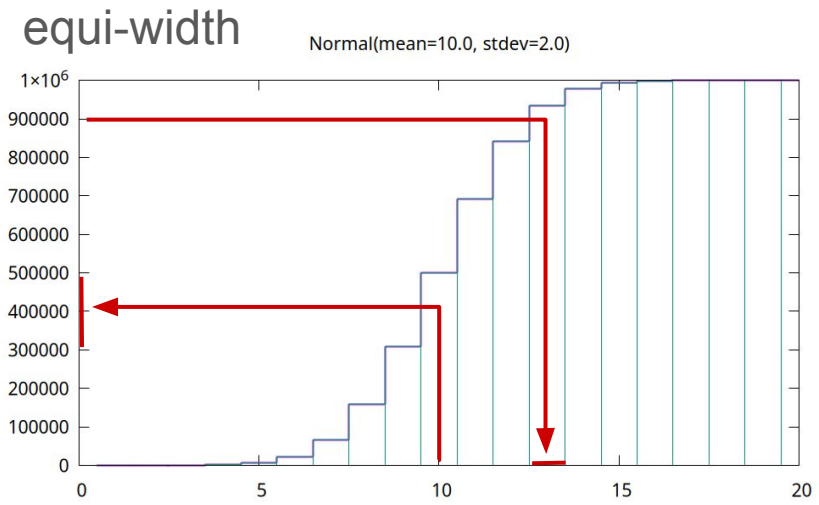
CDF

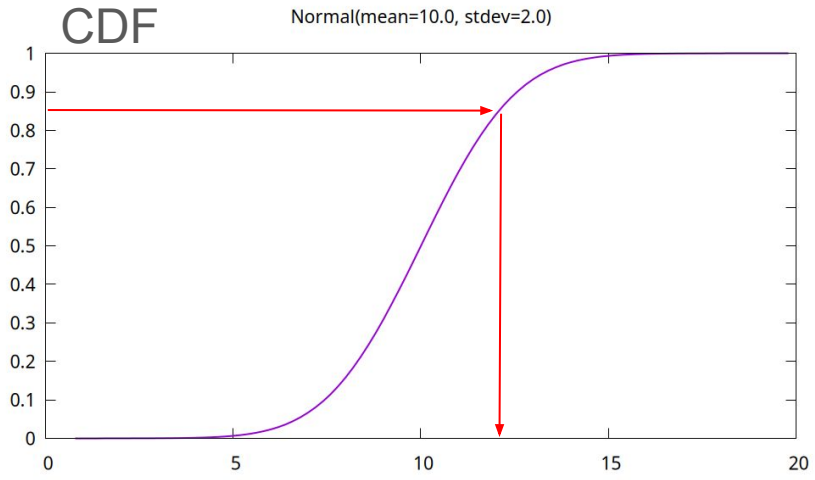


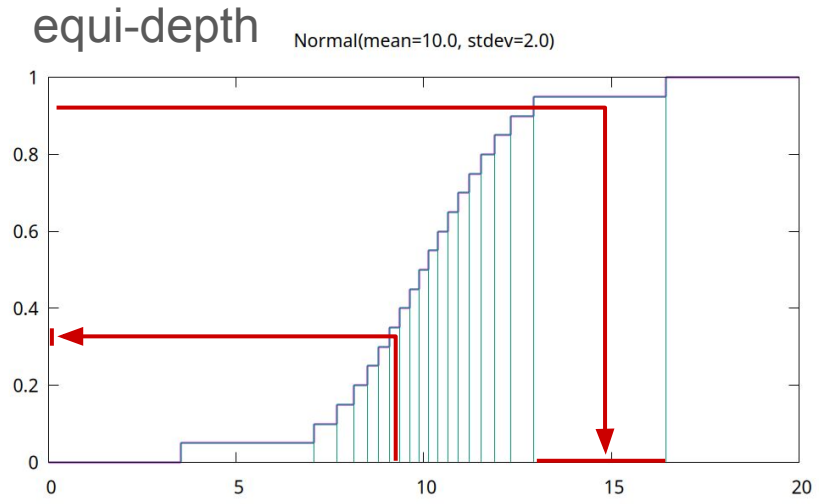
equi-depth



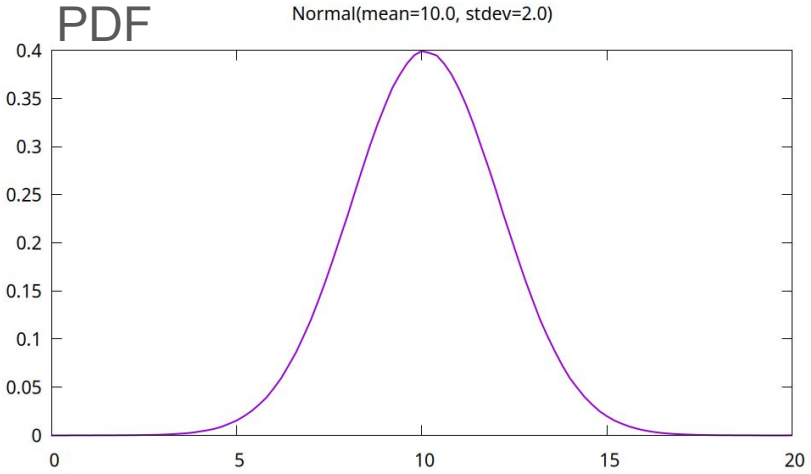

PDF
Normal(mean=10.0, stdev=2.0)
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
5
10
15
20


equi-width
Normal(mean=10.0, stdev=2.0)
1×10^6
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
0
5
10
15
20

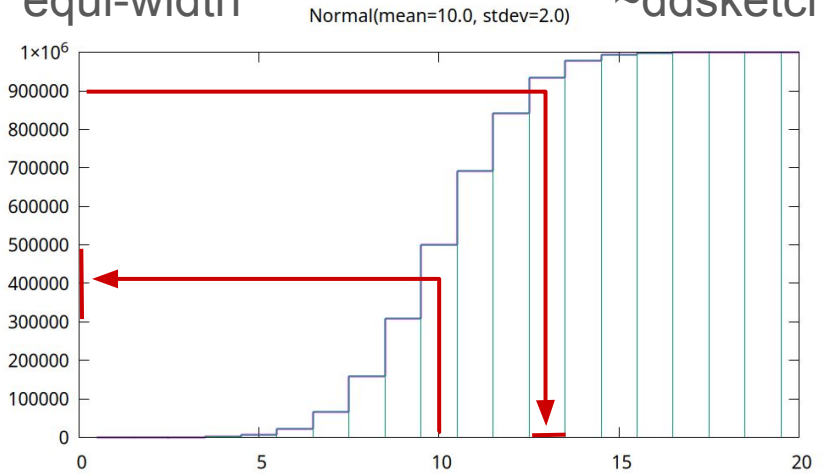

CDF
Normal(mean=10.0, stdev=2.0)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5
10
15
20


equi-depth
Normal(mean=10.0, stdev=2.0)
1
0.8
0.6
0.4
0.2
0
0
5
10
15
20

PDF

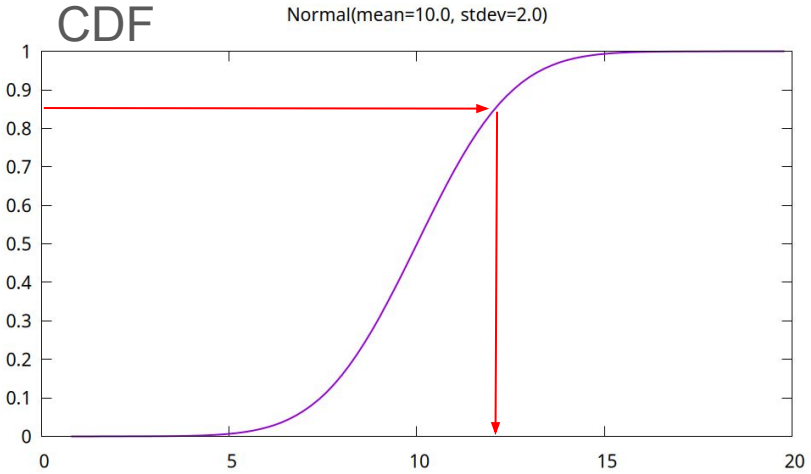


equi-width

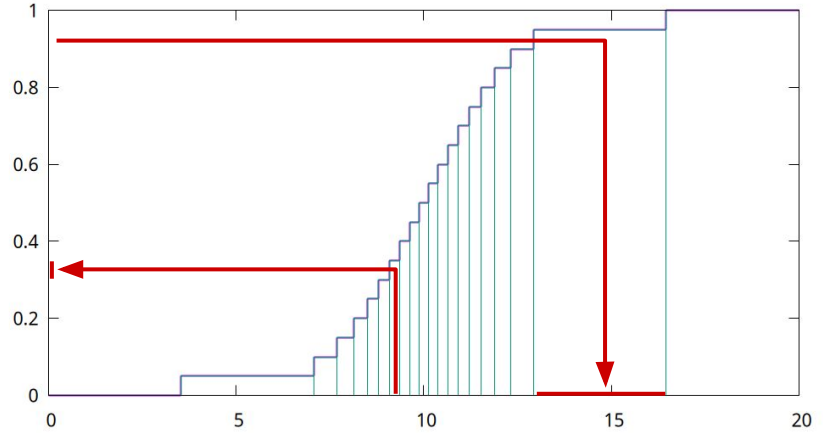


~ddsketch

CDF



equi-depth



~t-digest

Microsoft

t-digest

t-digest [https://github.com/tvondra/tdigest]

Microsoft

- 2019 paper by Ted Dunning, Otmar Ertl

  Computing Extremely Accurate Quantiles Using t-Digests
  https://github.com/tdunning/t-digest

- basic idea: clustering / centroids
  - represent data as "centroids" - pairs (value, count)
  - centroid ~ histogram bin
  - accumulate data until some space limit
  - merge centroids in a way that "maximizes" resolution

t-digest

Microsoft

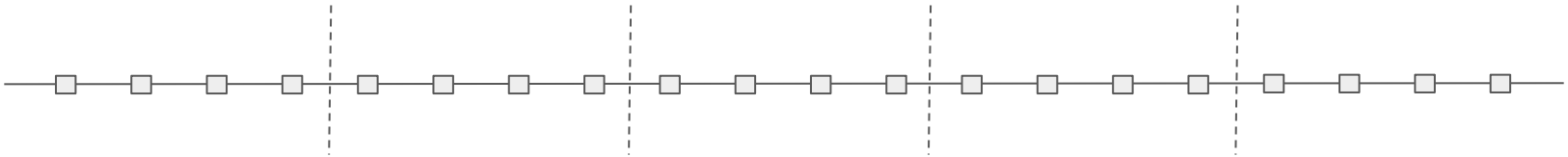
20 values, let's build 5 bins

in practice:
millions of points
hundreds of bins

t-digest

Microsoft

simple: equi-width bins

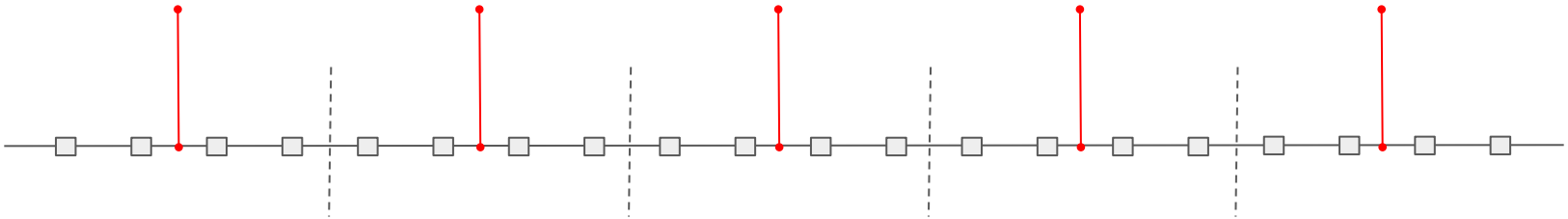


t-digest

Microsoft

simple: equi-width bins

(centroids are red)

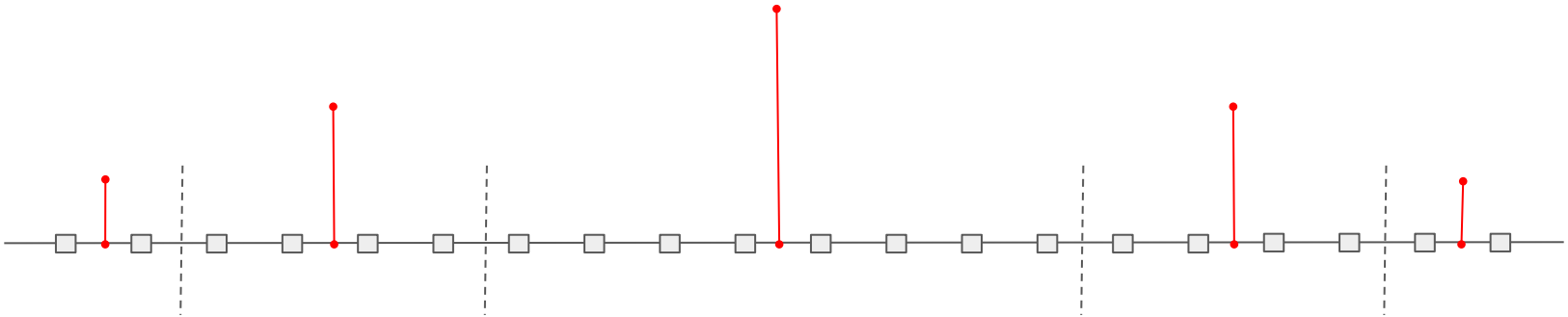


t-digest

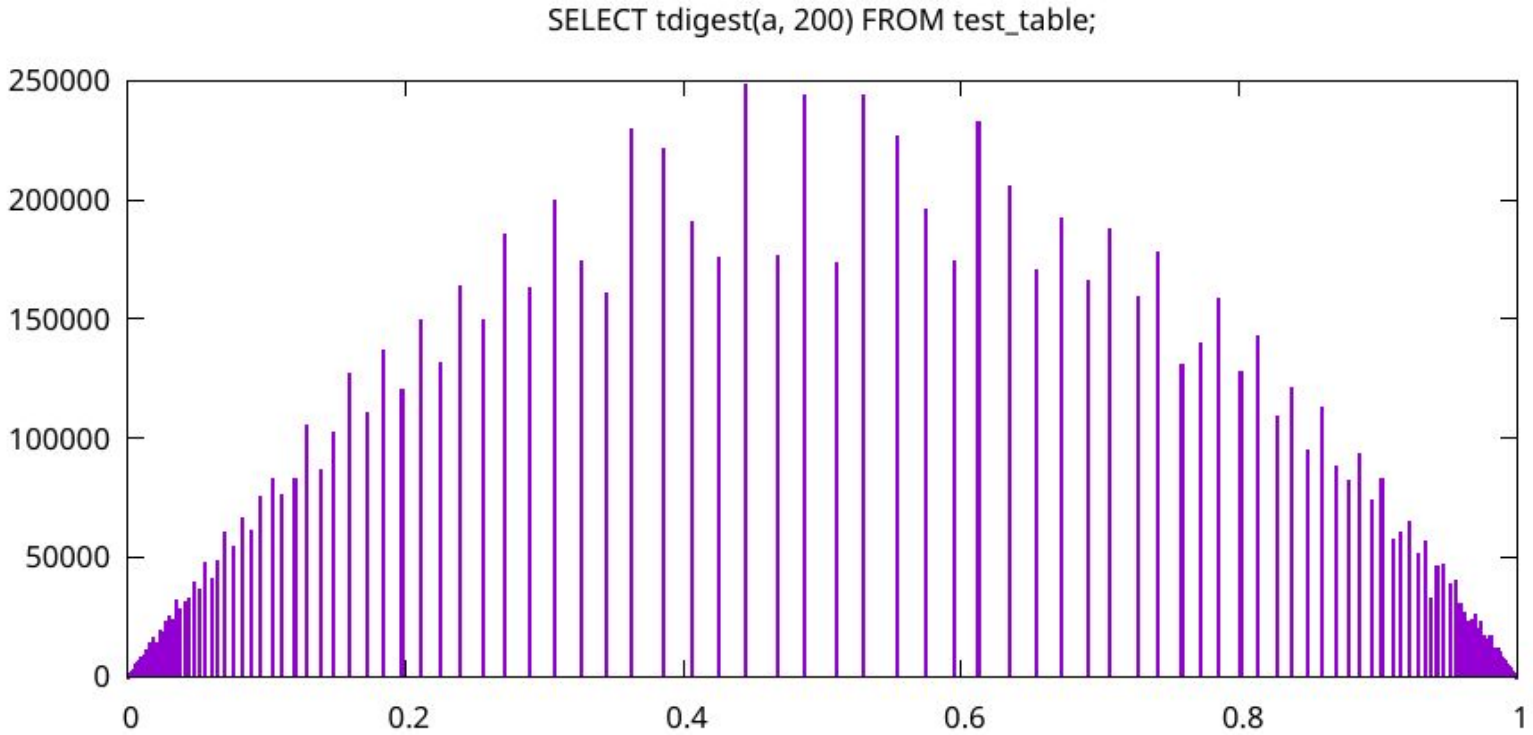
Microsoft

advanced: smaller bins on tails

(better accuracy ~0.0 and ~1.0)



t-digest - 10M values, uniform [0, 1]



t-digest example

Microsoft

```
SELECT req_method,
    tdigest_percentile(tdigest(req_time, 100), array[0.5, 0.95])
FROM request_timings GROUP BY req_method;

                                  QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate  (cost=204052.90..204065.40 rows=1000 width=36)
   Group Key: req_method
   ->  Seq Scan on request_timings  (cost=0.00..154053.60 rows=9999860 width=12)
(3 rows)

Time: 2565.322 ms (percentile_cont: 3693.776 ms)
```

t-digest example

```
SELECT req_method,
    tdigest_percentile_of(tdigest(req_time, 100), array[50.0, 95.0])
FROM request_timings GROUP BY req_method;

                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Finalize GroupAggregate  (cost=92616.86..93138.30 rows=1000 width=36)
   Group Key: req_method
   ->  Gather Merge  (cost=92616.86..93095.80 rows=4000 width=36)
         Workers Planned: 4
         ->  Sort  (cost=91616.80..91619.30 rows=1000 width=36)
               Sort Key: req_method
               ->  Partial HashAggregate  (cost=91554.48..91566.98 rows=1000 width=36)
                     Group Key: req_method
                     ->  Parallel Seq Scan on request_timings  (cost=0.00..79054.65 rows=2499965 width=12)
(9 rows)

Time: 967.469 ms (percentile_cont: 2869.319 ms)
```

t-digest

Microsoft

- result depends on data ordering
  - can be a problem for "weird" data (e.g. perfectly correlated / sorted)
  - more a batch/reporting problem, not for user requests
- result depends on parallelism
  - workers see different subsets of data on each run
  - different partial results in workers, similar to ordering
- inherently not deterministic
  - unless you fix the ordering / parallelism
  - surprising, makes testing harder

t-digest

Microsoft

- doesn't have reliable error guarantees
  - "rank error" guarantee for strongly ordered case
  - breaks due to "weak ordering" of merged digest (overlapping centroids)
  - but merging also allows parallelism, precalculation, incremental builds
  - works well in practice, though (good empirical results)
- "rank error" is not the right error
  - it's pretty much the "opposite" of what users need

Microsoft

ddsketch

ddsketch [https://github.com/tvondra/ddsketch]

Microsoft

- 2019 VLDB paper by Datadog (real-time monitoring SaaS)

  DDSketch: A Fast and Fully-Mergeable Quantile Sketch with Relative-Error Guarantees

  https://www.vldb.org/pvldb/vol12/p2195-masson.pdf

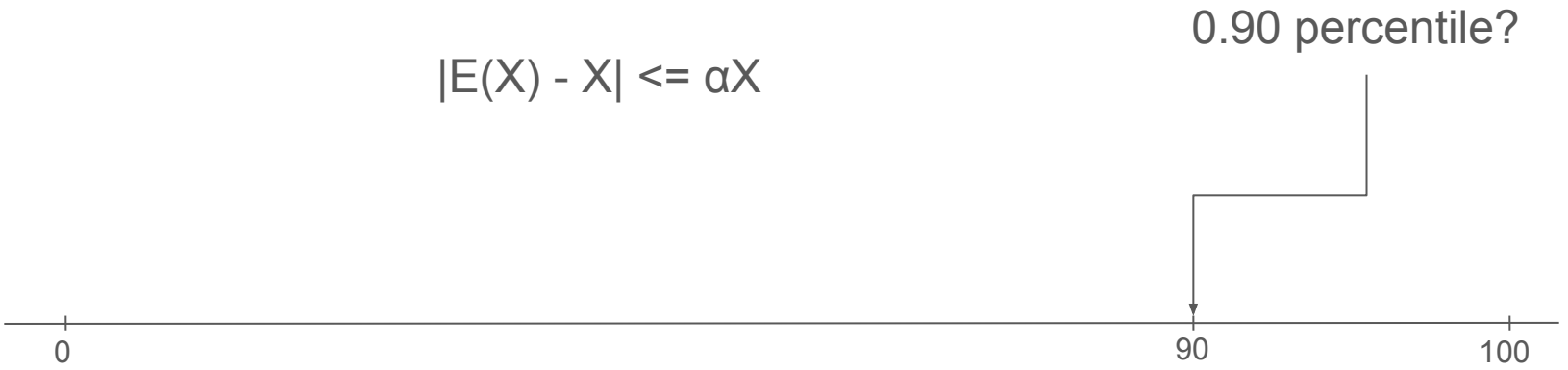
- DDSketch = Distributed Distribution Sketch
  - ... but also the company is called "Datadog" ;-)
- very different approach (from t-digest)
  - limits relative error (this guides the design)
  - deterministic, buckets determined by α-accuracy

ddsketch

goal:

- non-overlapping bins
- guarantee that

$$|E(X) - X| <= \alpha X$$

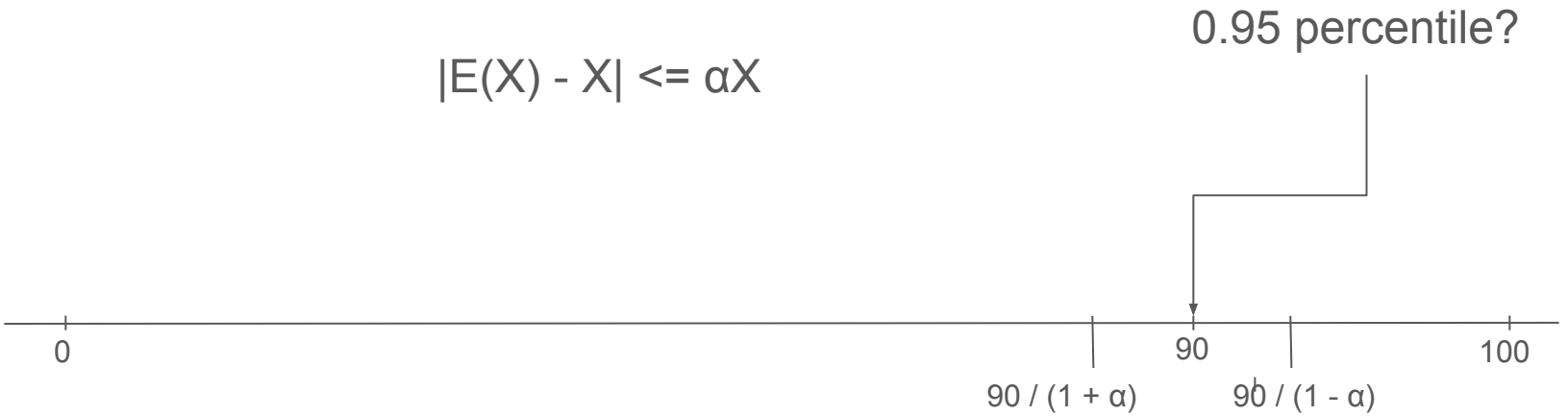


ddsketch

goal:

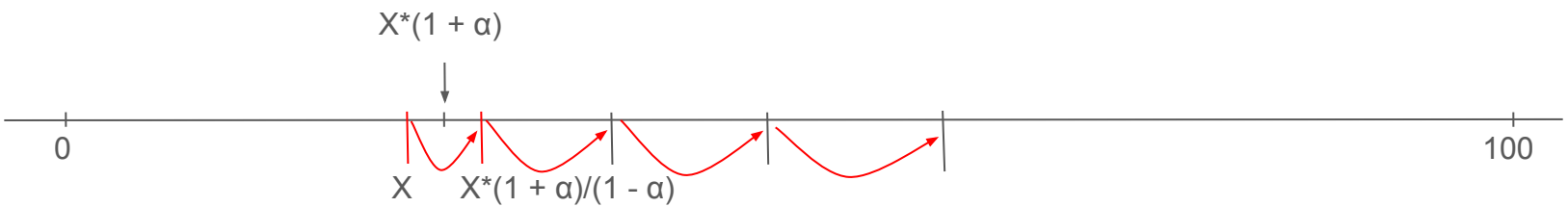
- non-overlapping bins
- guarantee that

$$|E(X) - X| <= \alpha X$$



ddsketch

$$|E(X) - X| <= \alpha X$$

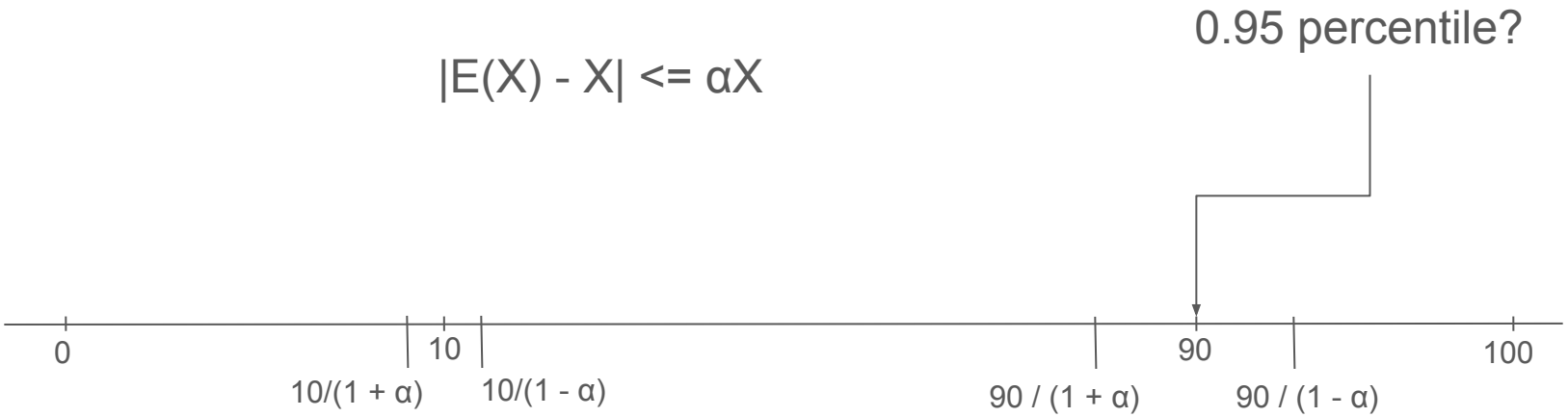


ddsketch

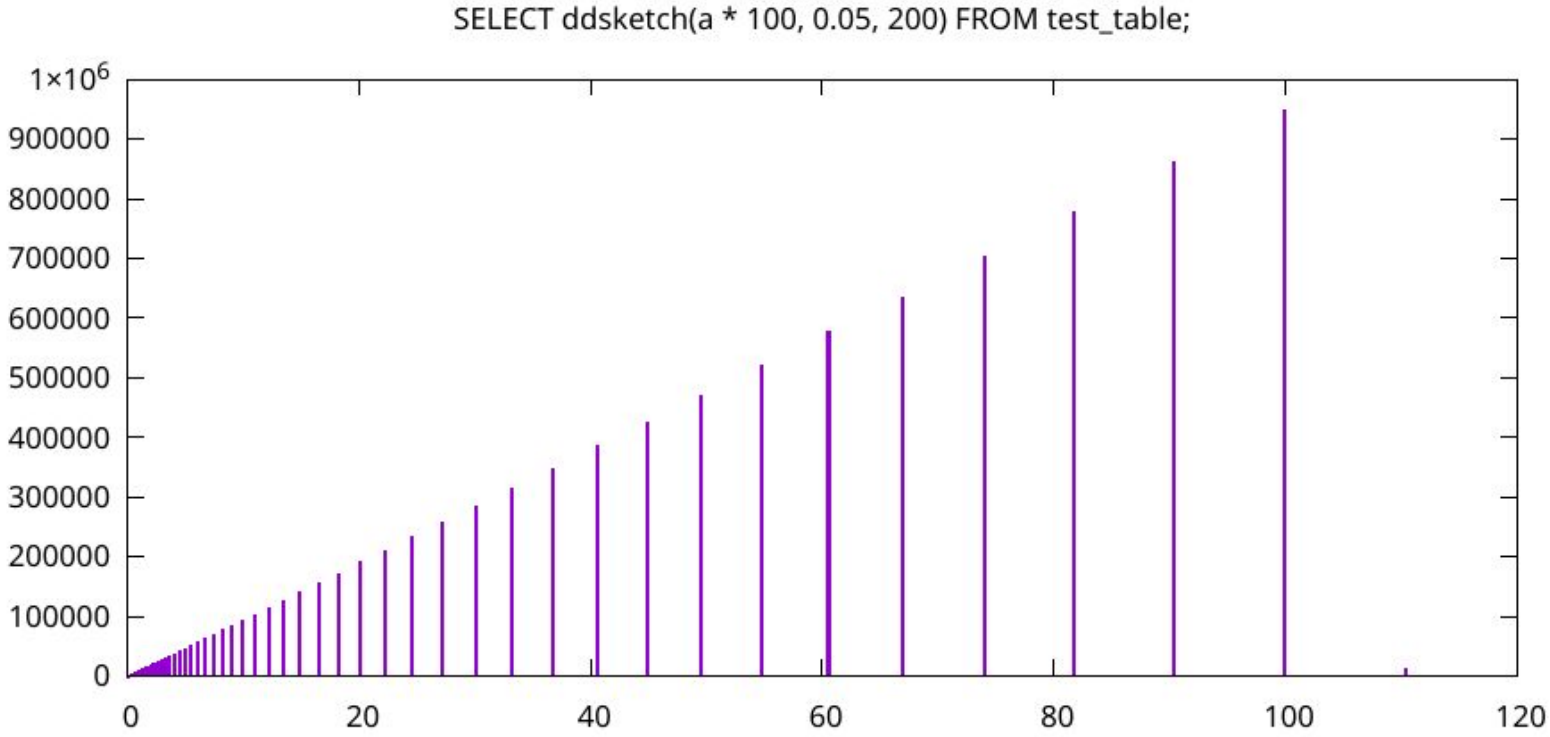
goal:

- non-overlapping bins
- guarantee that

$$|E(X) - X| <= \alpha X$$



ddsketch - 10M values, uniform [0, 100]



ddsketch example

Microsoft

```
SELECT req_method,
    ddsketch_percentile(ddsketch(req_time, 0.05, 1024), array[0.5, 0.95])
FROM request_timings GROUP BY req_method;

                                  QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate  (cost=204052.90..204065.40 rows=1000 width=36)
   Group Key: req_method
   ->  Seq Scan on request_timings  (cost=0.00..154053.60 rows=9999860 width=12)
(3 rows)

Time: 1932.321 ms (percentile_cont: 3693.776 ms, t-digest: 2565.322 ms)
```

ddsketch example

Microsoft

```
SELECT req_method,
    ddsketch_percentile(ddsketch(req_time, 0.05, 1024), array[0.5, 0.95])
FROM request_timings GROUP BY req_method;

                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Finalize GroupAggregate  (cost=92616.86..93138.30 rows=1000 width=36)
   Group Key: req_method
   ->  Gather Merge  (cost=92616.86..93095.80 rows=4000 width=36)
         Workers Planned: 4
         ->  Sort  (cost=91616.80..91619.30 rows=1000 width=36)
               Sort Key: req_method
               ->  Partial HashAggregate  (cost=91554.48..91566.98 rows=1000 width=36)
                     Group Key: req_method
                     ->  Parallel Seq Scan on request_timings  (cost=0.00..79054.65 rows=2499965 width=12)
(9 rows)

Time: 490.612 ms (percentile_cont: 2869.319 ms, t-digest: 967.469 ms)
```

Microsoft

precalculation

precalculation

- OLAP - precalculate at fine granularity
  - still a significant compression
  - aggregate these precalculated results (fast)
- API response times
  - pre-aggregate per-minute digests (incremental)
  - fast dashboards with 1h windows + drill down
- also in distributed / monitoring systems
  - remote system aggregates into sketch
  - transmits small sketch instead of full raw event stream

precalculation: ddsketch

Microsoft

```
CREATE TABLE request_sketches AS
SELECT req_method,
    ddsketch(req_time, 0.05, 1024) AS s
FROM request_timings GROUP BY req_method;
```

```
                              List of relations
 Schema |      Name         | Type  | Owner | Persistence | Access method |  Size
--------+-------------------+-------+-------+-------------+---------------+---------
 public | request_sketches  | table | user  | permanent   | heap          | 1544 kB
 public | request_timings   | table | user  | permanent   | heap          | 422 MB
(2 rows)
```

precalculation: ddsketch

```
SELECT
    ddsketch_percentile(ddsketch(s), array[0.5, 0.95])
FROM request_sketches WHERE req_method IN (1,2,3);

                                QUERY PLAN
-------------------------------------------------------------------------
 Aggregate  (cost=205.76..205.77 rows=1 width=32)
   ->  Seq Scan on request_sketches  (cost=0.00..205.75 rows=3 width=1260)
         Filter: (req_method = ANY ('{1,2,3}'::integer[]))
(3 rows)


 Execution Time: 0.932 ms
```

Microsoft

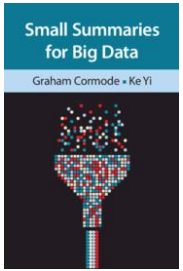
conclusions

conclusions

Microsoft

- sketches are a great & general idea!
  - Who says you can't design a sketch for other things?
  - hyperloglog, count-min, omnisketch
- powerful approach
  - estimate is often good enough (it's a trade-off)
  - merging allows parallelizing / precalculating
- t-digest/ddsketch
  - CDF approximation = form of a histogram
  - accuracy guarantees
  - can answer other questions (averages, percentile-of, ...)

sources



- Small Summaries for Big Data / Graham Cormode, Ke Yi
  https://www.cambridge.org/core/books/small-summaries-for-big-data/B41310C236A3D3574C273C42B71F35A4
- Apache DataSketches
  https://datasketches.apache.org/
- count-min sketch
  https://en.wikipedia.org/wiki/Count%E2%80%93min_sketch
- DDSketch: A fast and fully-mergeable quantile sketch with relative-error guarantees
  https://arxiv.org/abs/1908.10693
- UDDSketch: Accurate Tracking of Quantiles in Data Streams
  https://arxiv.org/abs/2004.08604
- Computing Extremely Accurate Quantiles Using t-Digests
  https://arxiv.org/abs/1902.04023

Q & A

Office Hours

- private discussions about Postgres stuff
- send me an email with info in advance
- Tuesday, 16:00 - 17:00 CET (Prague)

  ... or propose a different time
- 30-minute slots by default, may be extended
- Google Meet ...
- https://vondra.me/about/