Fast Approximations for Analyzing Ten Trillion Cells

Filip Buruiana (filipb@google.com)

Reimar Hofmann (reimar.hofmann@hs-karlsruhe.de)

Outline of the Talk

- Interactive analysis at AdSpam @ Google
- Trade off accuracy for speed
- Sampling - challenges, accuracy, results
- Data sketch inspired approach
  for reducing memory usage

AdSpam Team @ Google

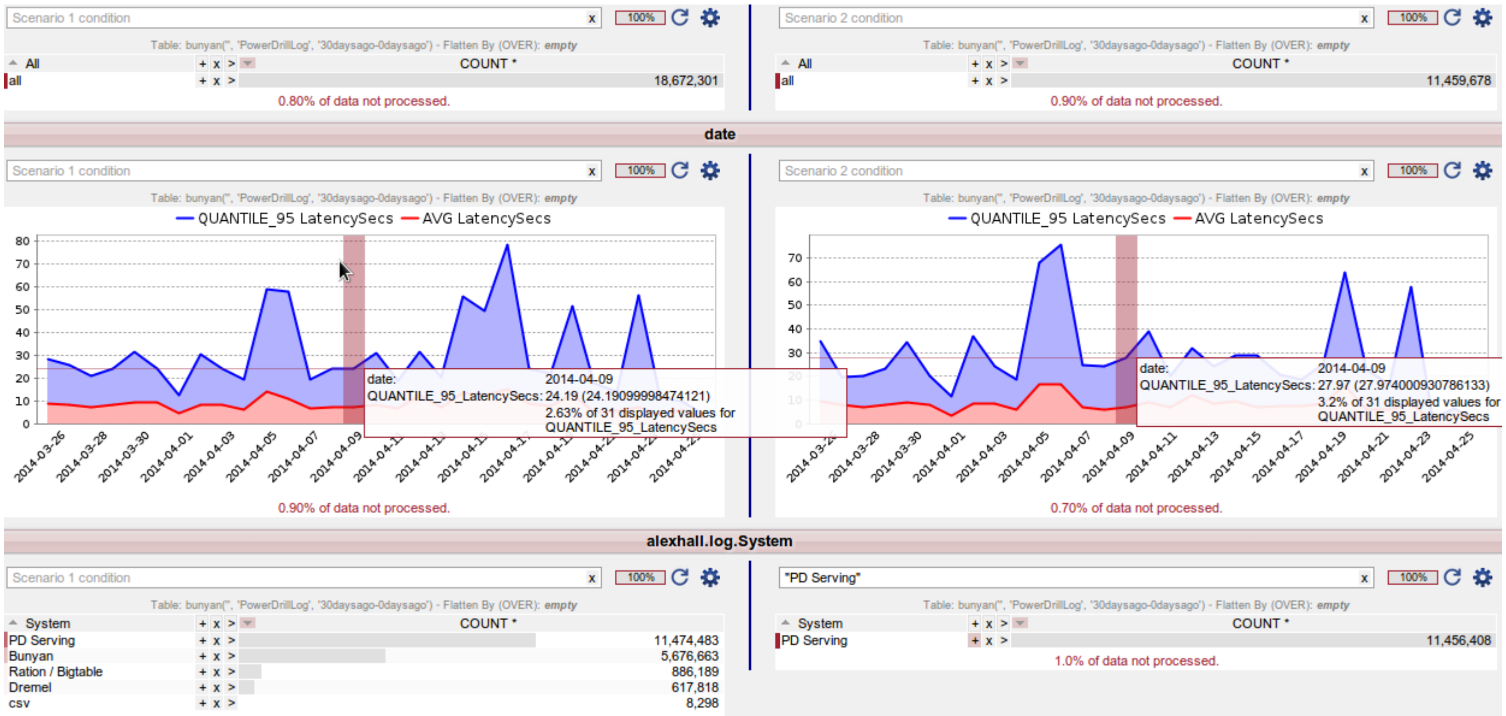
- Click police: filter "invalid clicks" & charge only for organic traffic

- Typical data sets: billions of records

- Typical use-cases:
  - Ops - manually review suspicious clicks
  - Eng - research new filter ideas

PowerDrill: Workflows

- one UI interaction ⇒ multiple SQL queries

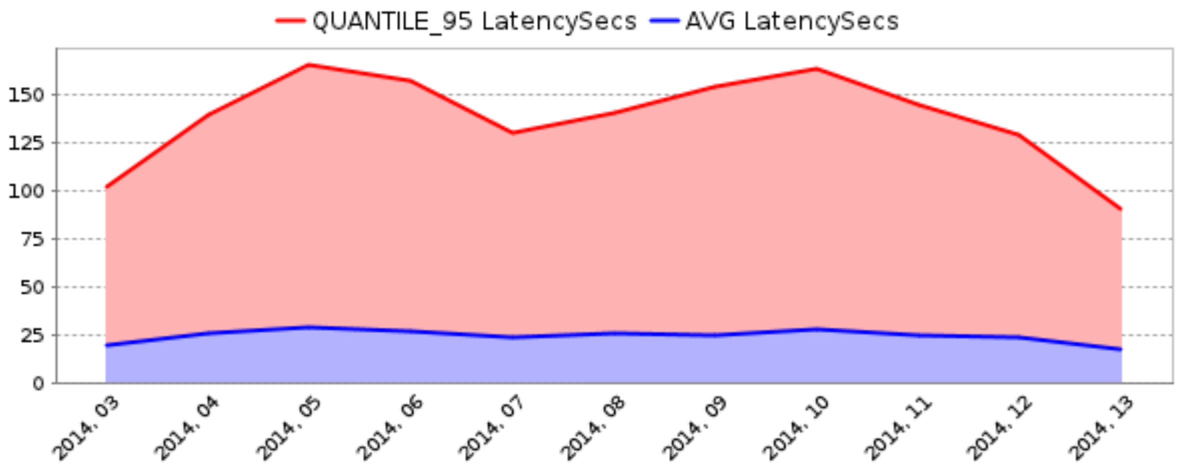


- data discovery, finding patterns ⇒ many interactions

* Displayed data is from PowerDrill usage logs

Interactive: Motivation

weekly latency, overall (in seconds)



red: 95%-quantile
blue: average

Waiting for results:

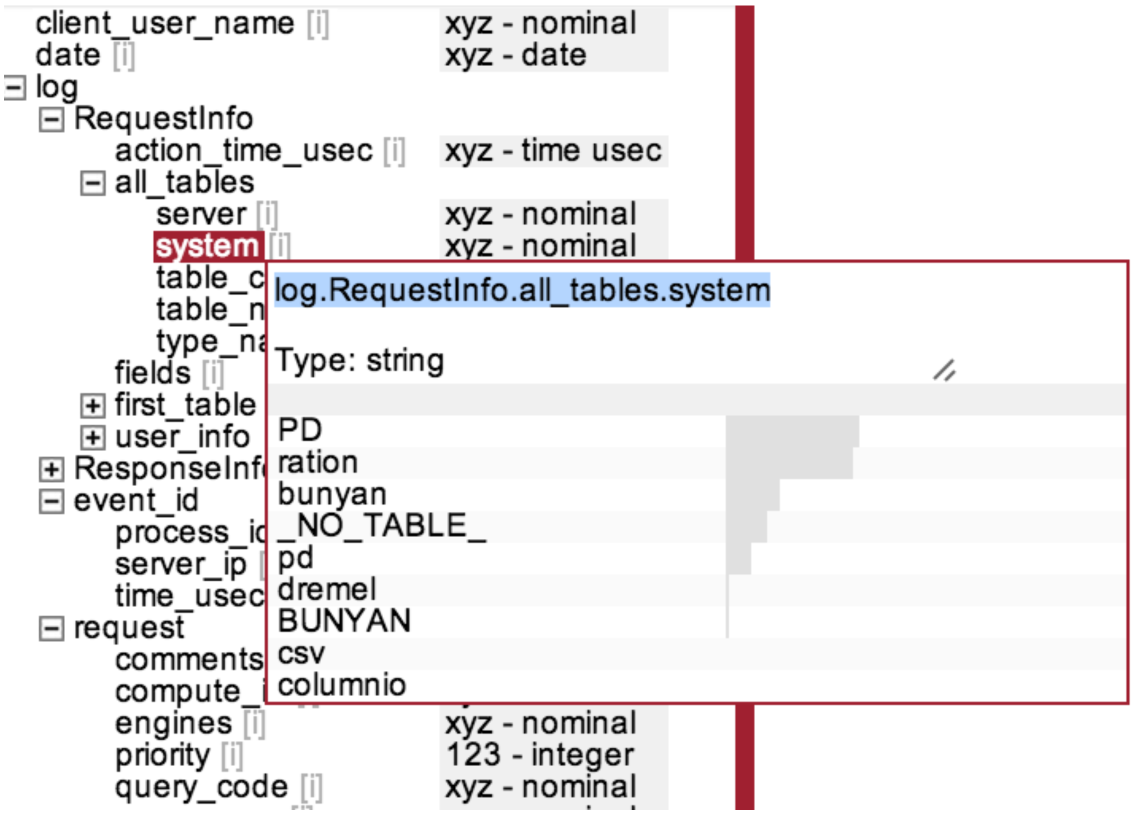
- costs money
- users get bored

Trade off accuracy for speed

- Some workflows: exact (e.g. reporting)
- Others: approx is fine
  - Analyze longer-time trends
  - “Good enough” intermediate results ⇒ quick decisions
  - Hints for the user: what to explore next

UI features enabled by faster queries

- Histogram in field info box

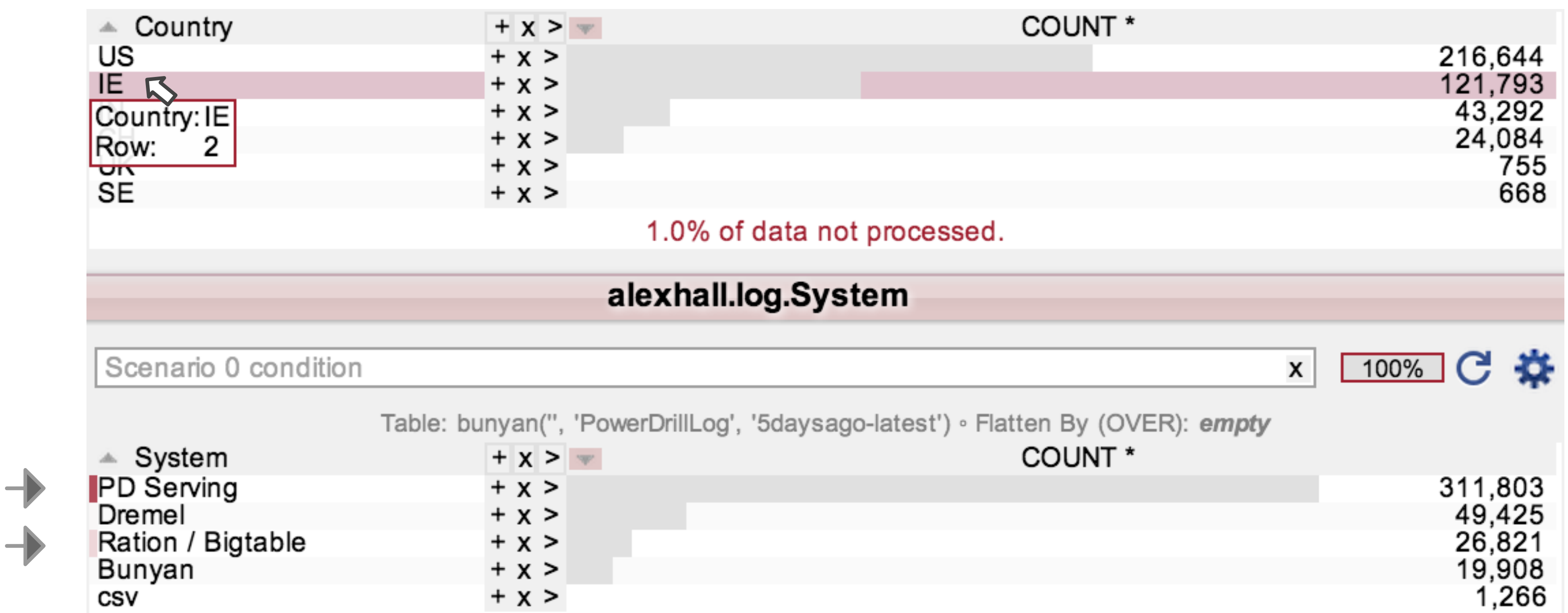


* Schema is from PowerDrill usage logs

UI features enabled by faster queries

- ## Cross chart highlighting

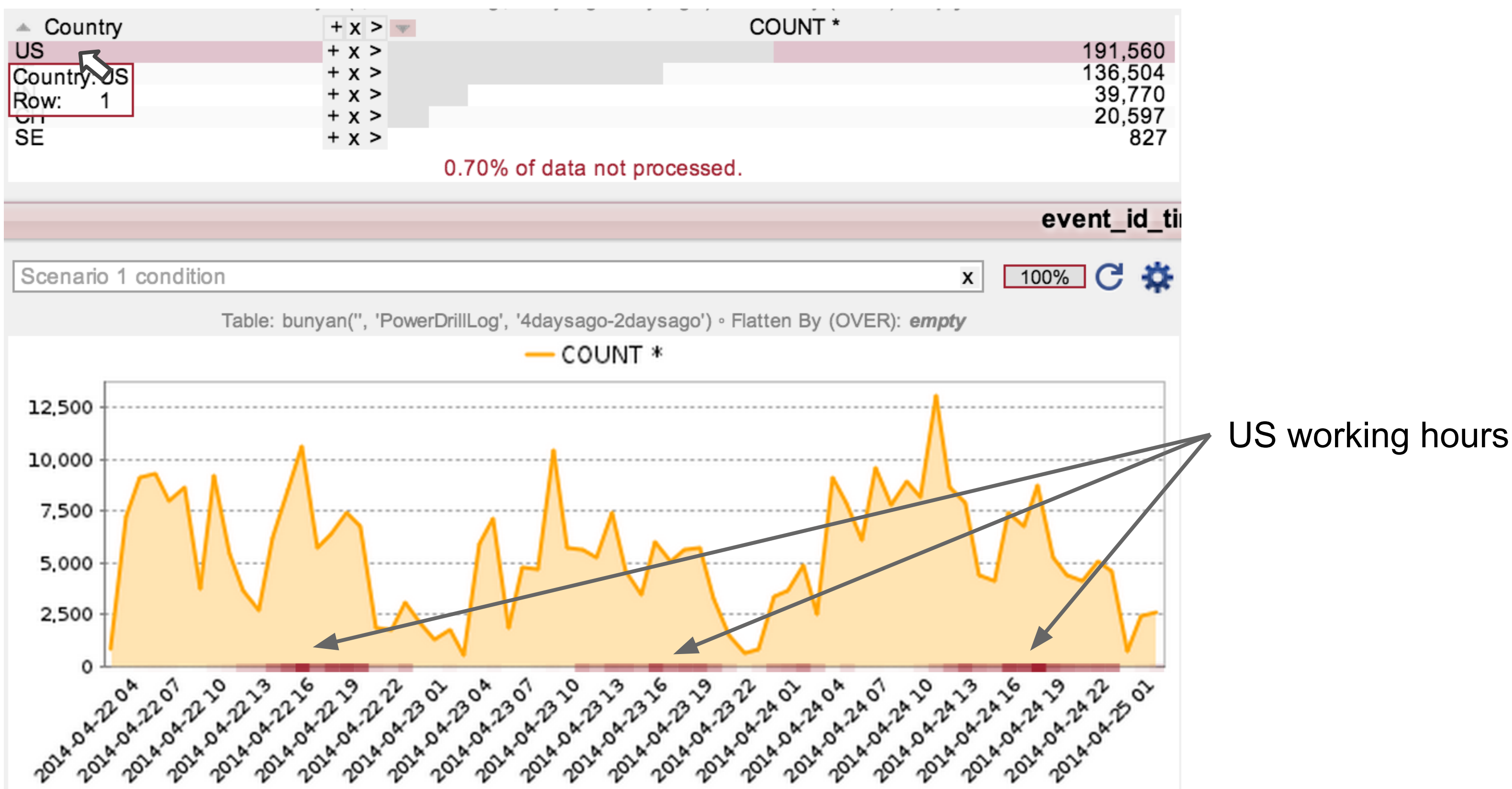
  Hovering over a value in a chart marks positively correlated values in other charts



* Displayed data is from PowerDrill usage logs

UI features enabled by faster queries

- Cross chart highlighting



* Displayed data is from PowerDrill usage logs

Sampling

Memory reduction with maximum query flexibility:

- Particularities of our system
- Predicting accuracy
- Performance results

Hierarchical relational data

PowerDrill contains hierarchical data

Sampling happens on the top level

→ Consider when estimating sampling accuracy of lower level fields

→ Consider when designing data model for new data

Sampling: Dealing with Inaccuracy

"Accurate" → P(***relative error*** **< 10%) > 90%**

"Inaccurate" → greyed out

Country	+	x	>	COUNT *	SUM LatencySecs x	COUNT_DISTINCT_UI user_name x
US	+	x	>	1,562,462	32,782,116.86	≈ 458.00
CH	+	x	>	1,096,746	29,432,247.75	≈ 107.00
IE	+	x	>	971,371	9,594,336.50	≈ 66.00
IN	+	x	>	271,772	4,288,256.49	≈ 89.00
UK	+	x	>	3,504	69,778.92	≈ 24.00
SG	+	x	>	≈ 1,977	≈ 26,974.70	≈ 5.00
AU	+	x	>	≈ 1,702	≈ 45,936.39	≈ 14.00
JP	+	x	>	≈ 1,376	≈ 10,295.03	≈ 10.00
CA	+	x	>	≈ 951	≈ 18,277.54	≈ 3.00
SE	+	x	>	≈ 676	≈ 2,948.02	≈ 2.00

Accuracy depends on

- the number of records supporting the estimate, and
- the variance of the data column.

* Displayed data is from PowerDrill usage logs

Sampling: Predicting Accuracy

Evaluation on real queries (10% sample):

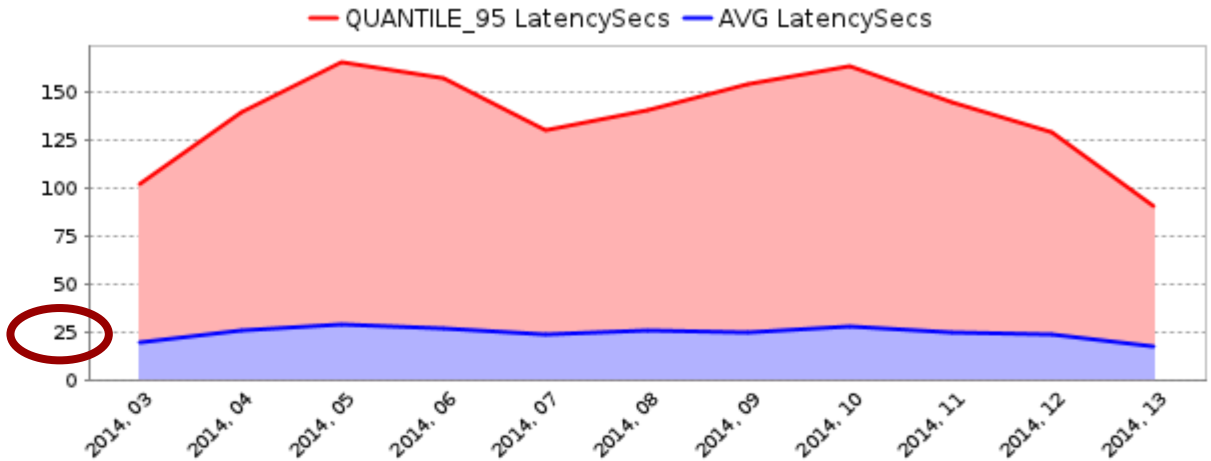
- avoid computing sample variance → heuristic

	Accuracy Prediction based on				True accuracy
	Sample Variance		**Heuristic**		
	classified as ...	correctly	**classified as ...**	correctly	
… “accurate”	**67%**	92%	**42%**	90%	**75%**
… “inaccurate”	33%	61%	58%	36%	25%

based on 30 K cells of queries from the query log

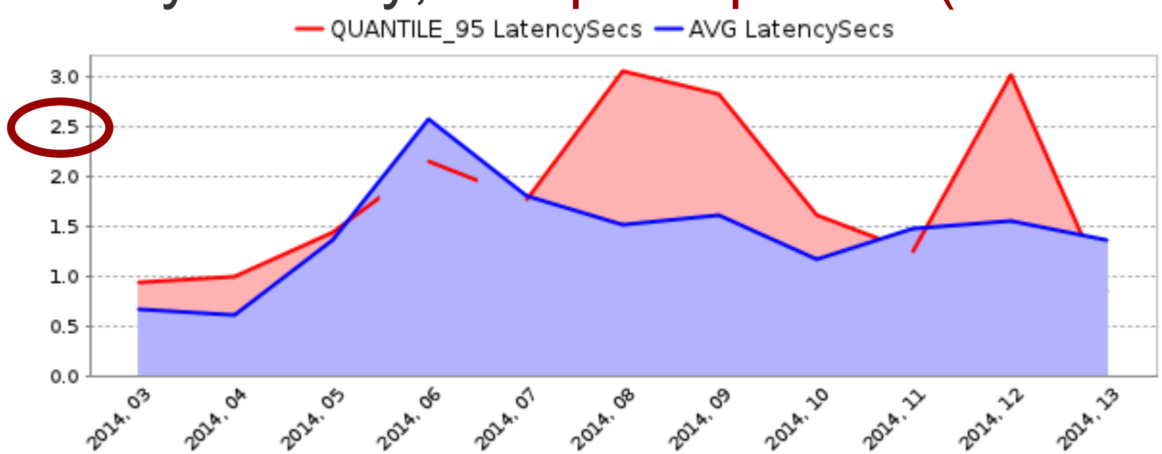
Sampling: Success Metrics

weekly latency, overall (in seconds)



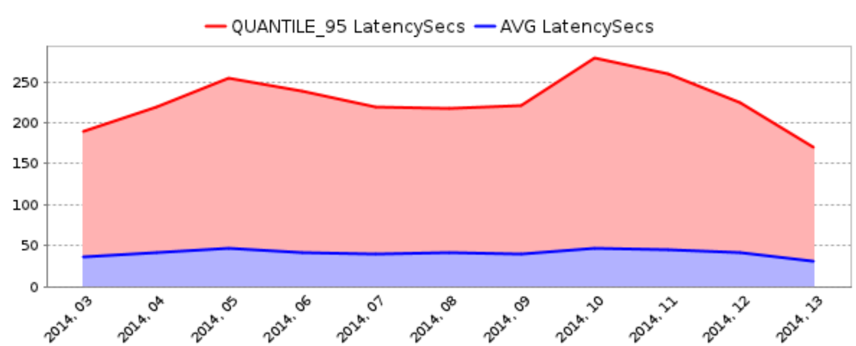
red: 95%-quantile
blue: average

weekly latency, sampled queries (17% of overall)

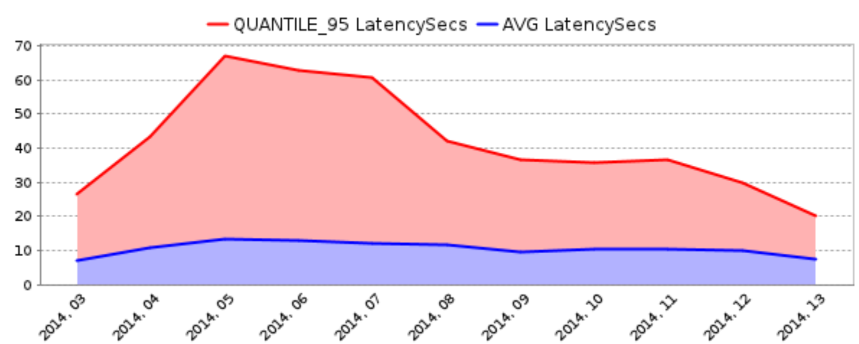


Bottleneck: Disk loads

- Low latency variance for sampled: better cache hits (# items on disk: 27% for sampled vs. 34% non-sampled)
- Data >> available RAM ⇒ memory + disk



latency of queries that touch disk



latency of queries served entirely from memory

- Disk loads = high latency

Expensive fields

- 2% of overall queries: 70% of all memory ⇒ affects all queries
- Top-K queries with expensive fields

```
SELECT SearchQuery as Query,
          Count() as Count
FROM Table
GROUP BY Query
ORDER BY Count
LIMIT 10
```

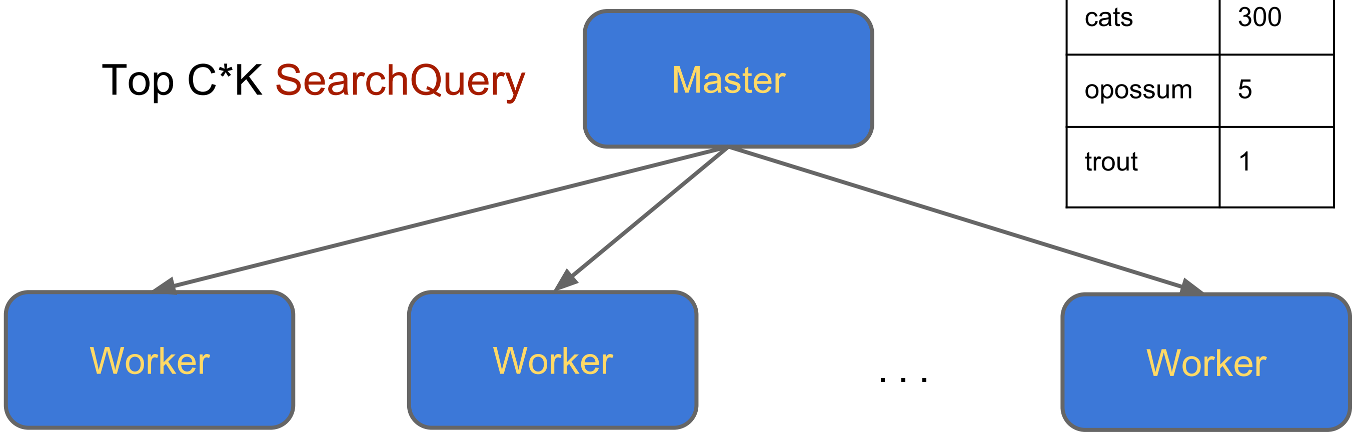
many unique long string values ⇒ **450 MB** memory / server

- 10% of memory caches for only one field!

Execution of Top-K Queries

Example: K = 1, C = 3

Search Query	Count
dogs	**320**
cats	300
opossum	5
trout	1



Search Query	Count
cats	100
dogs	90
opossum	2

C*K string lookups

Search Query	Count
dogs	140
cats	100
trout	1

C*K string lookups

Search Query	Count
cats	100
dogs	90
opossum	3

C*K string lookups

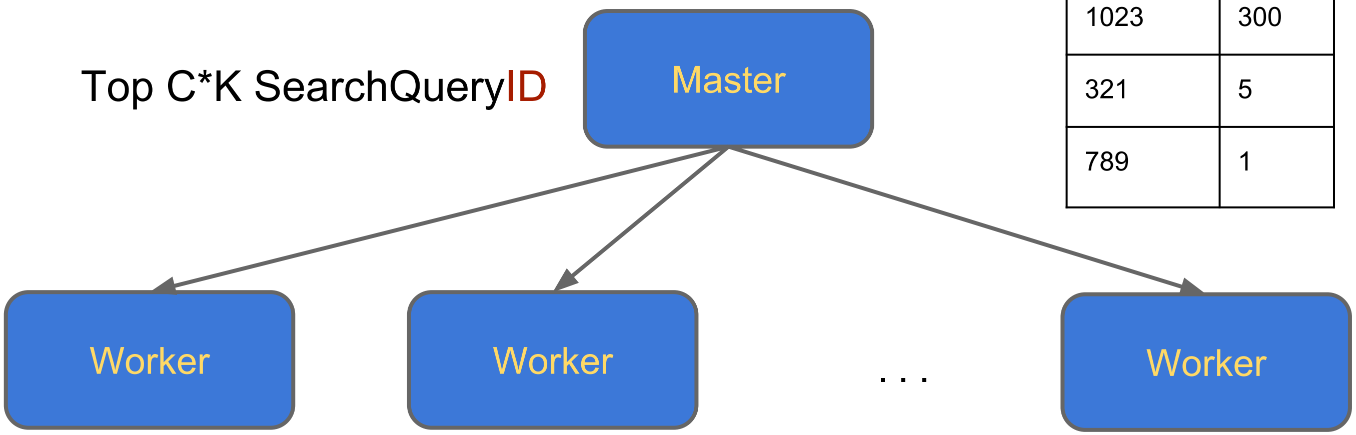
String values ⇒ IDs

Example: K = 1, C = 3

Search QueryID	Count
609	**320**
1023	300
321	5
789	1

K string lookups

Top C*K SearchQueryID



Search QueryID	Count
1023	100
609	90
321	2

0 string lookups

Search QueryID	Count
609	140
1023	100
789	1

0 string lookups

Search QueryID	Count
1023	100
609	90
321	3

0 string lookups

Reverse Lookup Service

Search QueryID	Search Query
321	opossum
609	dogs
789	trout
1023	cats

Reverse lookup dictionary

billions of distinct values

most frequent items: memory
the rest: distributed file system

- Globally consistent IDs: hard
- ID space needs to be used efficiently

Global IDs = Hashes

Use predefined hash function

- globally consistent
- controllable size of IDs
- collisions can lead to ambiguous, “lifted” results

SearchQuery	Count
‘cats’ *or* ‘opossums’	900
‘dogs’	899

Collision Probability

Example:

some fields → D ≈ 3 bn distinct values

hash length 32 bits → N ≈ 4 bn distinct IDs

(more: implementation penalty)

$$1 - \left(1 - \frac{1}{N}\right)^{D}$$

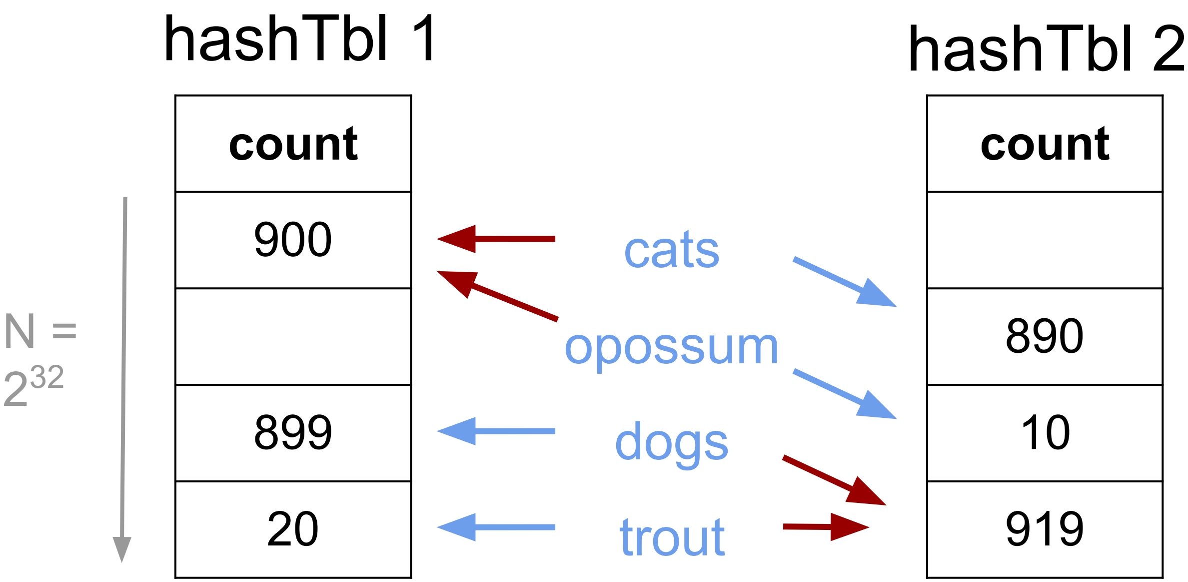
≈ **50%**

collision probability

⇒ Can't afford to keep collisions rare

Background: The Count-Min-Sketch

Use multiple redundant hash tables:



'cats': **min**(900, 890)
'opossum': **min**(900, 10)

- Results accurate unless item collides in both hash tables
- Well known error bounds
- Known tradeoff between number of hashes and hash size.

CM-Sketch-inspired approach

Idea:
Two data columns with two different hashes of *SearchQuery*

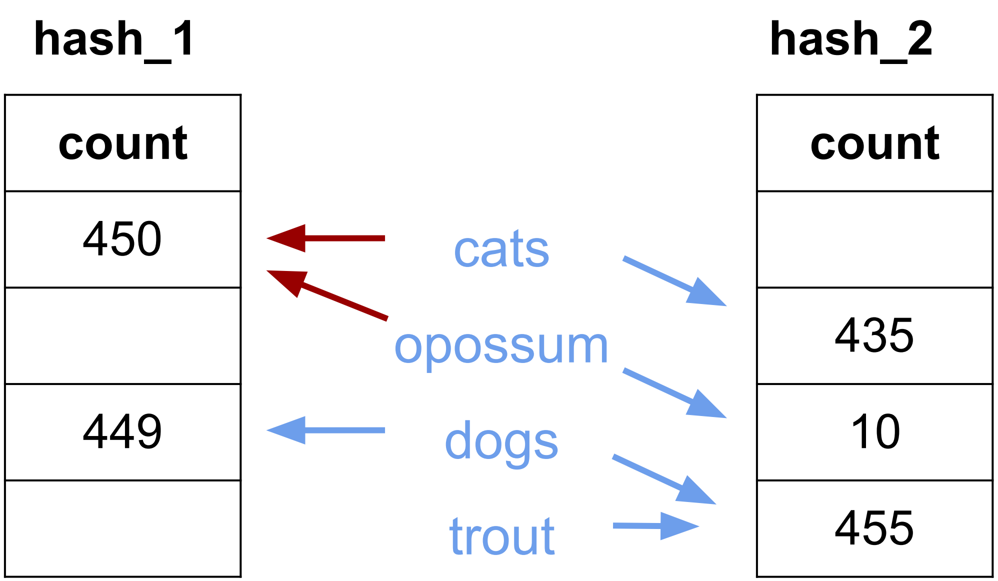
Count min sketch	**Our approach**
Hash function as index into (dense) tables.	Hash function as encoding for a data column
WHERE-conditions can not be applied once table is filled	WHERE-conditions from other fields can be applied

- Error bounds of CM-sketches apply
- Collisions can be resolved
- Optimal hash sizes are different

But: Redundancy wastes memory.

CM-Sketch-inspired approach

- Half of the data is encoded with hash_1, other half with hash_2.
- No memory is wasted.
- Most collisions can be resolved approximately:



‘Cat’ ↔ ‘opossum’:

count(‘cat) ≈ 450 + 435 - 10

count(‘opossum’) ≈ 2 · 10

Accuracy

- At least as good as CM-sketch with one hash table.
Example: (31 bit hash for field *SearchQuery*)

  $$Lift := Count_{est} - Count_{true}$$

  $$E(Lift) = 7,\ P(Lift > 70) < 10\%,\ P(Lift > 700) < 1\%$$

  $Count > 70000$ for all TOP 100 SearchQueries

  $\Rightarrow$ P(*relative error* > 1%) < 1%

- Similarly low error rates for other fields and queries.
- Benefit of splitting data:
Resolve ambiguities in reverse lookup.

Memory Benefits

SearchQuery: 325 GB $\rightarrow$ 90 GB (3.6x reduction)

other field: 95 GB $\rightarrow$ 60 GB (1.6x reduction)

⇒ **Store ~ 3x more items in memory**

⇒ Speed up all queries

Conclusion

- Approximations help to get faster / simpler
- Enable a new level of usability concepts
- Accuracy can be predicted