Data profiling with Apache Calcite

DataWorks Summit 2017

SAN JOSE, USA

2017/06/14

Julian Hyde

@julianhyde

SQL
Query planning
Query federation
OLAP
Streaming
Hadoop

ASF member
Original author of Apache Calcite
PMC Apache Arrow, Drill, Eagle, Kylin

Overview

Apache Calcite

Motivating problem: Automatically designing summary tables

What is data profiling?

Naive profiling algorithm

Improving the algorithm using sketches, parallelism, information theory

Applying data profiling to other problems

Apache Calcite

Apache top-level project since October, 2015

Query planning framework

- ➢ Relational algebra, rewrite rules
- ➢ Cost model & statistics
- ➢ Federation via adapters
- ➢ Extensible

Packaging

- ➢ Library
- ➢ Optional SQL parser, JDBC server
- ➢ Community-authored rules, adapters

Used by

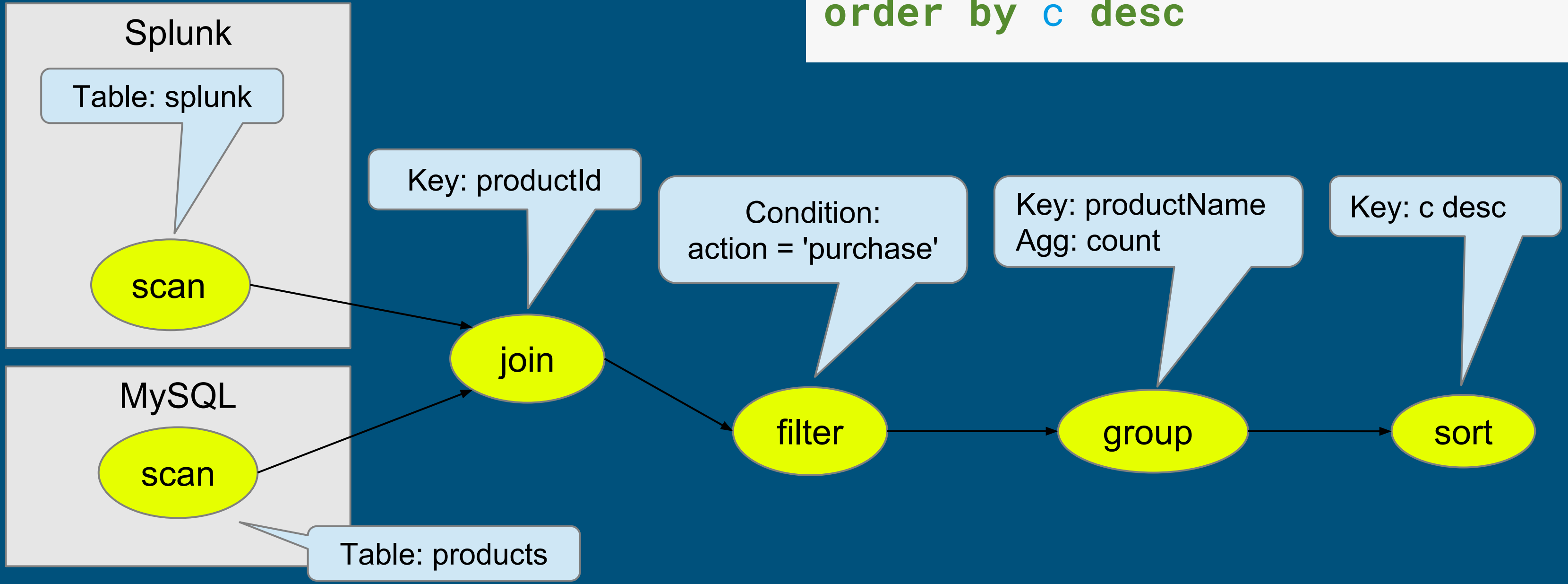
Hive, Apache Storm, NGDATA, Apache Drill, Apache Apex, druid, Apache Kylin, Apache Phoenix, Apache Solr, Apache Flink, samza, Cascading, Qubole, dremio, MapD, SQL Gremlin

Connects to

JDBC, Apache Spark, splunk, CSV, cassandra, elasticsearch, mongoDB

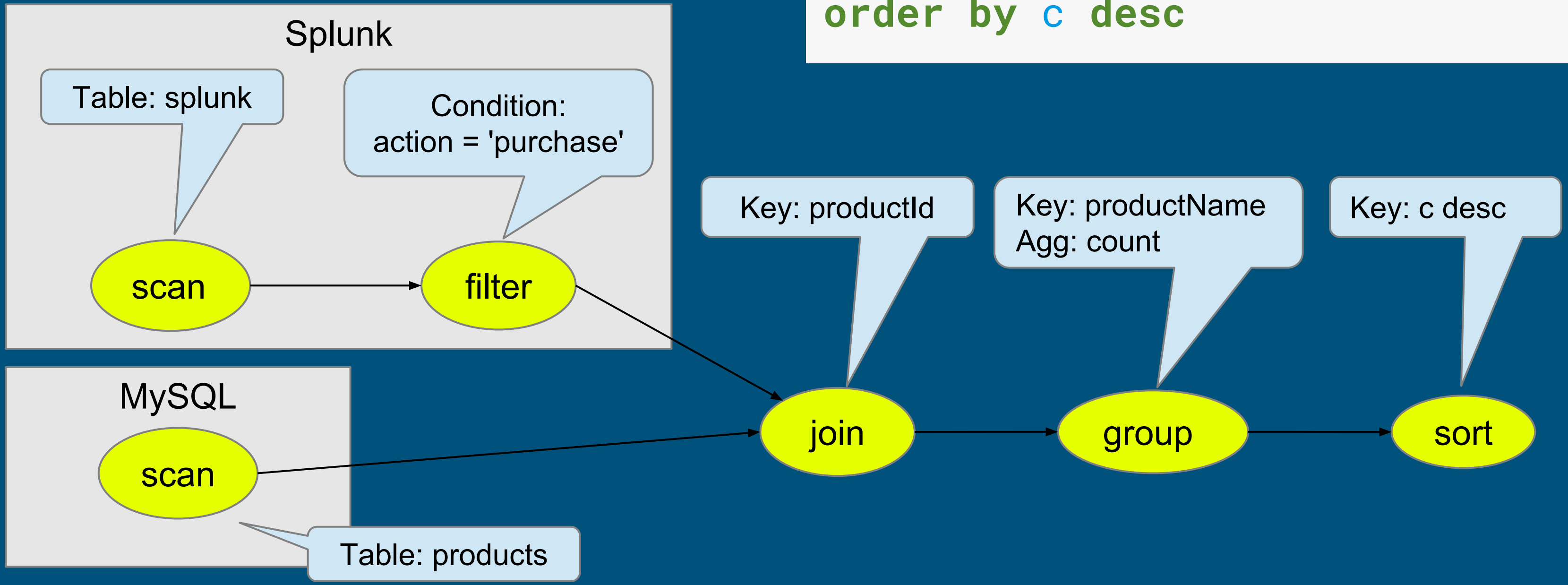
Planning queries

```
select p.productName, count(*) as c
from splunk.splunk as s
    join mysql.products as p
    on s.productId = p.productId
where s.action = 'purchase'
group by p.productName
order by c desc
```



Optimized query

```sql
select p.productName, count(*) as c
from splunk.splunk as s
    join mysql.products as p
    on s.productId = p.productId
where s.action = 'purchase'
group by p.productName
order by c desc
```



Optimizing queries

Problem

10 TB database, disk with 1 GB/s throughput, and a query that reads 1 TB data.

Solutions

1. **Sequential scan** Query takes 1,000s.
2. **Parallelize** Spread the data over 100 disks in 25 machines. Query takes 10s.
3. **Cache** Keep the data in memory. 2nd query: 10ms. 3rd query: 10s.
4. **Materialize** Summarize the data on disk. All queries: 100ms.
5. **Materialize + cache + adapt** As above, building summaries on demand.

Optimizing data

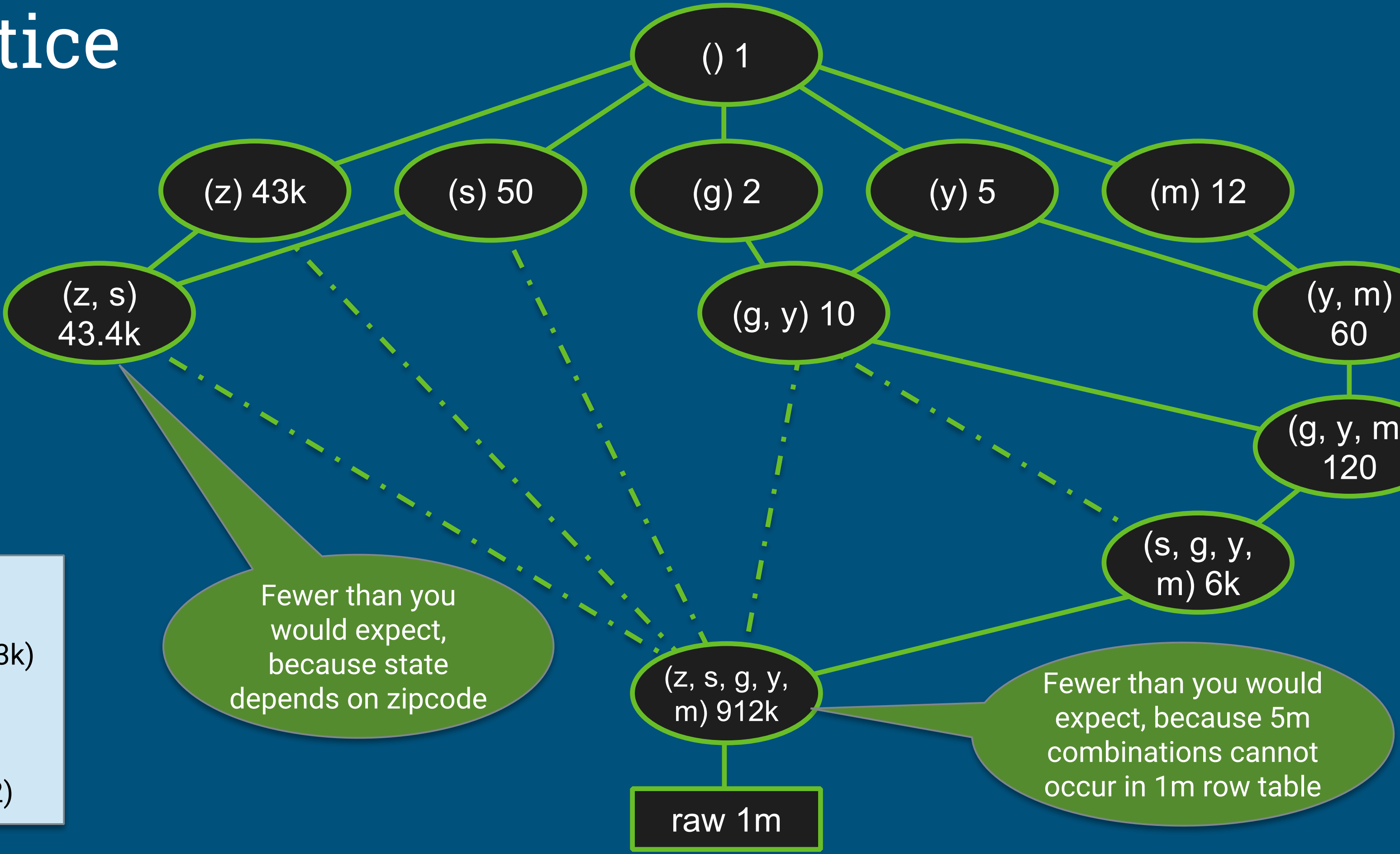
```
create materialized view EmpSummary as
select deptno, COUNT(*) as c, SUM(sal) as s
from Emp
group by deptno
```

A materialized view (“materialization”) is a table that contains the result of a query. The DBMS maintains it, and uses it to answer queries on other tables.

Challenges:

- **Design** Which materializations to create?
- **Populate** Load them with data
- **Maintain** Incrementally populate when data changes
- **Rewrite** Transparently rewrite queries to use materializations
- **Adapt** Design and populate new materializations, drop unused ones
- **Express** Need a rich algebra, to model how data is derived

Lattice



Key

z zipcode (43k)
s state (50)
g gender (2)
y year (5)
m month (12)

Algorithm: Design summary tables

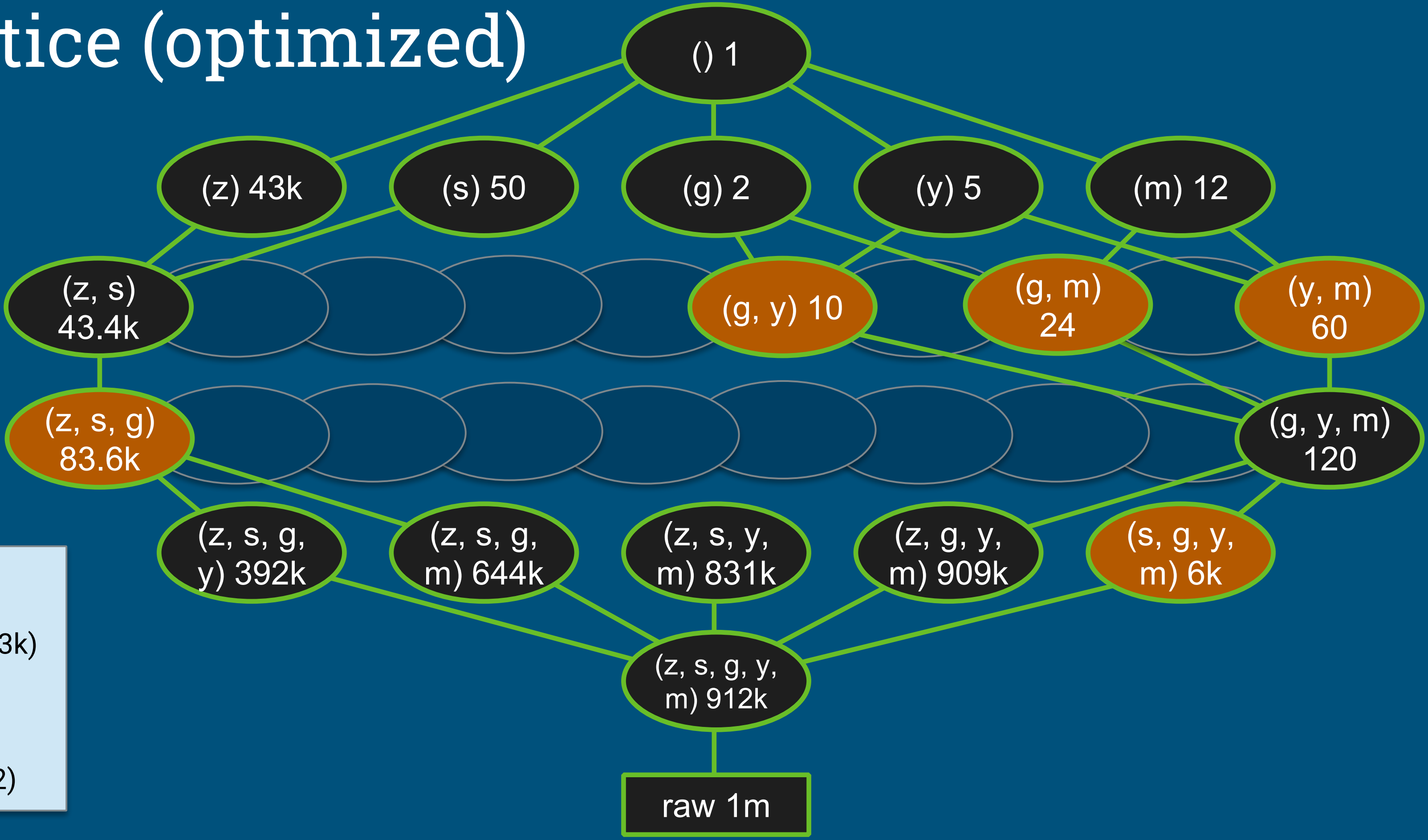
Given a database with 30 columns, 10M rows. Find X summary tables with under Y rows that improve query response time the most.

AdaptiveMonteCarlo algorithm [1]:

- Based on research [2]
- Greedy algorithm that takes a combination of summary tables and tries to find the table that yields the greatest cost/benefit improvement
- Models “benefit” of the table as query time saved over simulated query load
- The “cost” of a table is its size

[1] org.pentaho.aggdes.algorithm.impl.AdaptiveMonteCarloAlgorithm
[2] Harinarayan, Rajaraman, Ullman (1996). “Implementing data cubes efficiently”

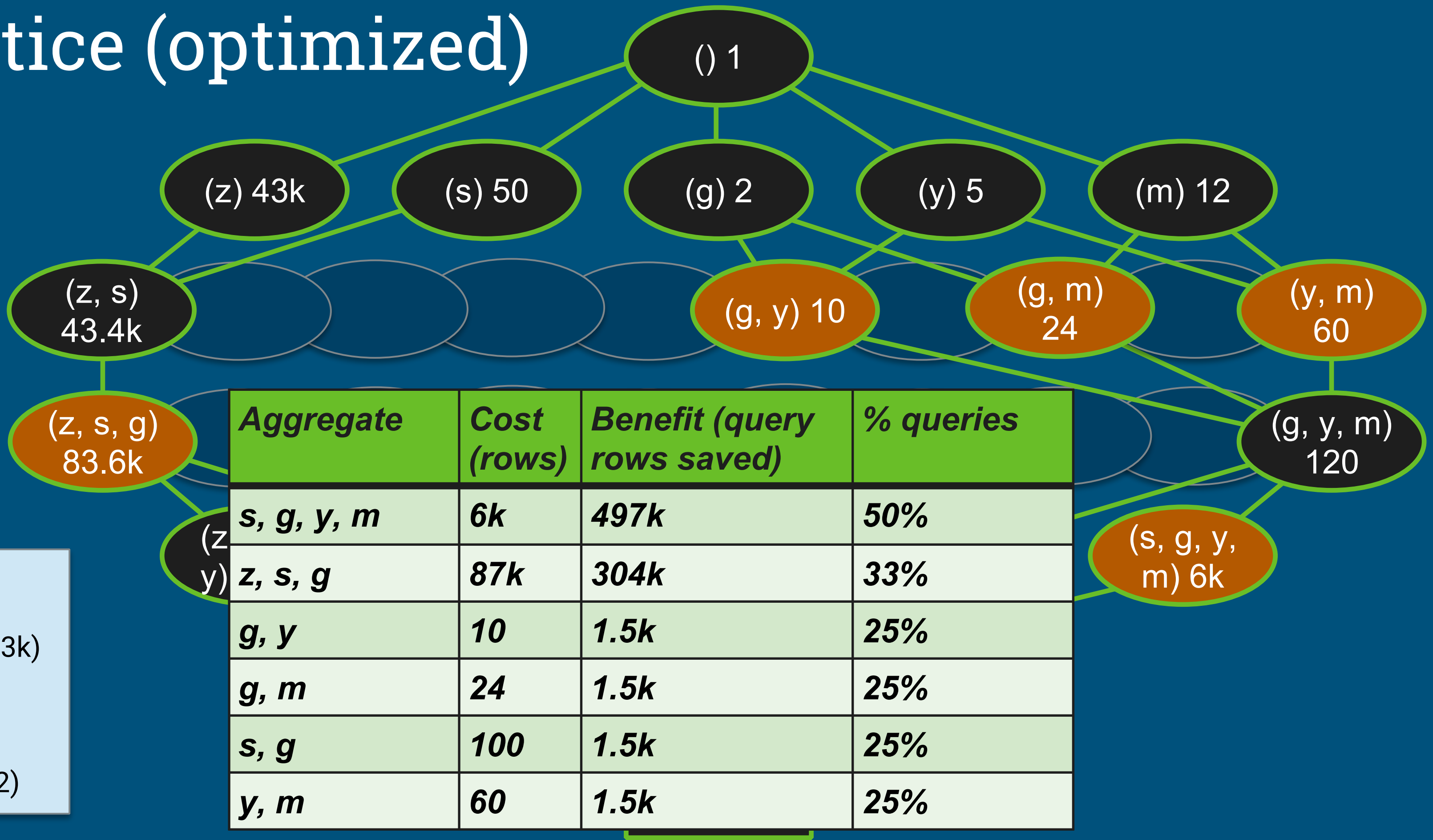
Lattice (optimized)



Key

z zipcode (43k)
s state (50)
g gender (2)
y year (5)
m month (12)

Lattice (optimized)



Aggregate	*Cost (rows)*	*Benefit (query rows saved)*	*% queries*
s, g, y, m	*6k*	*497k*	*50%*
z, s, g	*87k*	*304k*	*33%*
g, y	*10*	*1.5k*	*25%*
g, m	*24*	*1.5k*	*25%*
s, g	*100*	*1.5k*	*25%*
y, m	*60*	*1.5k*	*25%*

Key

z zipcode (43k)
s state (50)
g gender (2)
y year (5)
m month (12)

Data profiling

Algorithm needs `count(distinct a, b, ...)` for each combination of attributes:

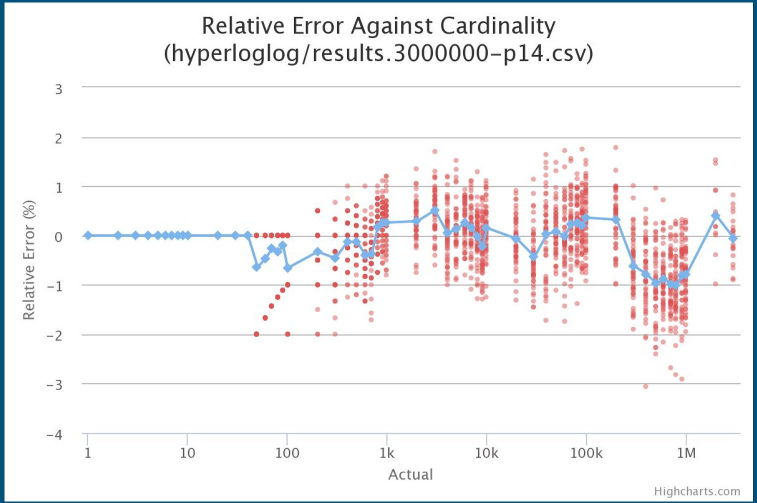
- Previous example had $2^5 = 32$ possible tables
- Schema with 30 attributes has 2^{30} (about 10^9) possible tables
- Algorithm considers a significant fraction of these
- Approximations are OK

Attempts to solve the profiling problem:

1. Compute each combination: scan, sort, unique, count; repeat 2^{30} times!
2. Sketches (HyperLogLog)
3. Sketches + parallelism + information theory (CALCITE-1616)

Sketches

HyperLogLog is an algorithm that computes approximate distinct count. It can estimate cardinalities of 10^9 with a typical error rate of 2%, using 1.5 kB of memory. [3][4]



With 16 MB memory per machine we can compute 10,000 combinations of attributes each pass.

So, we're down from 10^9 to 10^5 passes.

[3] Flajolet, Fusy, Gandouet, Meunier (2007). "Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm"
[4] https://github.com/mrjgreen/HyperLogLog

Combining probability & information theory

Given	Expected cardinality	Actual cardinality	Surprise
(gender): 2 (state): 50	(gender, state): 100.0	100	0.000
(month): 12 (zipcode): 43,000	(month, zipcode): 441,699.3	442,700	0.001
(state): 50 (zipcode): 43,000	(state, zipcode): 799,666.7	43,400	0.897
(state, zipcode): 43,400 (gender, state): 100 (gender, zipcode): 85,995	(gender, state, zipcode): 86,799 = min(86,799, 892,234, 892,228)	83,567	0.019

- Surprise = abs(actual - expected) / (actual + expected)
- E(card (x, y)) = n . (1 - ((n - 1) / n) ^ p) n = card (x) * card (y), p = row count

Algorithm

Three ways “surprise” can help:

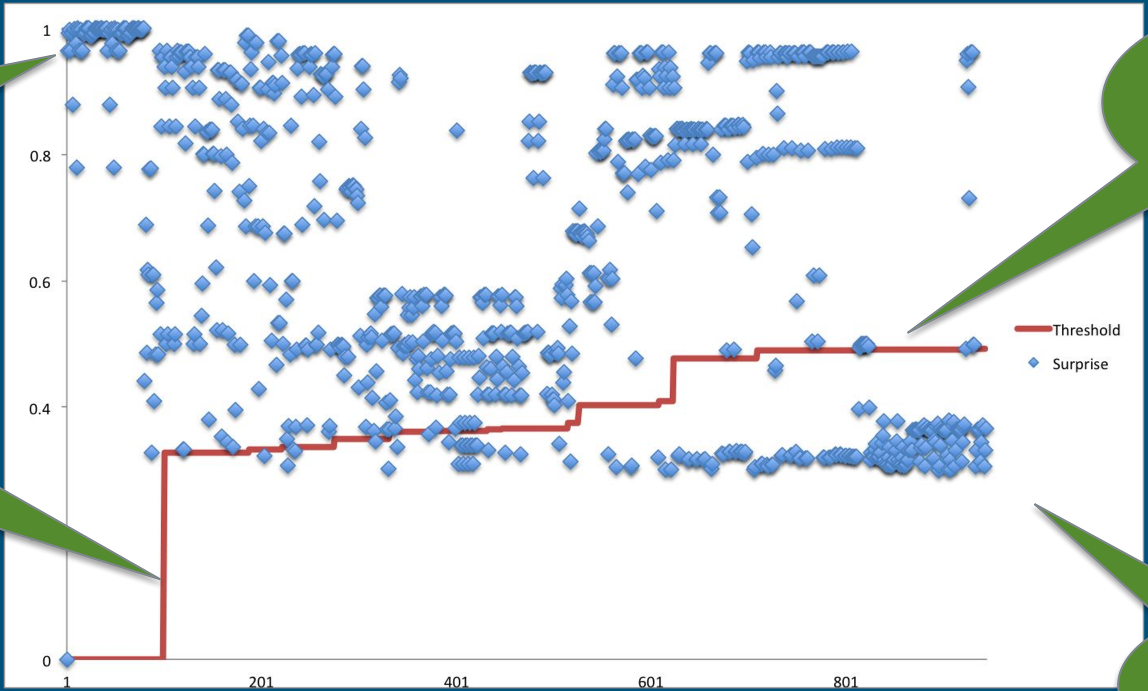
- If a cardinality is not surprising, we don’t need to store it -- we can derive it
- If a combination’s cardinality is not surprising, it is unlikely to have surprising children
- If we’re not seeing surprising results, it’s time to stop

```
surprise_threshold := 1
queue := {singleton combinations} // (a), (b), ...
while queue is not empty {
  batch := remove first 10,000 entries in queue
  compute cardinality of each combination in batch
  for each actual (computed) cardinality a {
    e := expected cardinality of combination
    s := surprise(a, e)
    if s > surprise_threshold {
      store combination and its cardinality
      add child combinations to queue // (x, a), (x, b), ...
    }
    increase surprise_threshold
  }
}
```

Algorithm progress and “surprise” threshold

Singleton combinations are have surprise = 1

Surprise thresold rises after we hve completed the first batch



Surprise threshold rises as algorithm progresses

Rejected as not sufficiently surprising

Progress of algorithm

Hierarchies considered harmful

Hierarchies are a feature of most OLAP systems

Does it makes sense to store (year, quarter, month, date) and roll up to (year, quarter)?

No -- algorithm can deduce hierarchies; less configuration means fewer mistakes

Summary optimizer naturally includes attributes that don't increase summary cardinality by much

Feel free to specify a "drill path" in slice & dice UI

True hierarchy

```
(year)
↑
(year, quarter)
↑
(year, quarter, month)
↑
(year, quarter, month, date)
```

Almost a hierarchy

```
(nation)
↑
(nation, state)
↑
(nation, state, zipcode)
```

Other applications of data profiling

Query optimization:

- Planners are poor at estimating selectivity of conditions after N-way join (especially on real data)
- New join-order benchmark: “Movies made by French directors tend to have French actors”
- Predict number of reducers in MapReduce & Spark

“Grokking” a data set

Identifying problems in normalization, partitioning, quality

Applications in machine learning?

Further improvements

- Build sketches in parallel
- Run algorithm in a distributed framework (Spark or MapReduce)
- Compute histograms
  - For example, Median age for male/female customers
- Seek out functional dependencies
  - Once you know FDs, a lot of cardinalities are no longer “surprising”
  - FDs occur in denormalized tables, e.g. star schemas
- Smarter criteria for stopping algorithm
- Skew/heavy hitters. Are some values much more frequent than others?
- Conditional cardinalities and functional dependencies
  - Does one partition of the data behave differently from others? (e.g. year=2005, state=LA)

Thank you!

https://issues.apache.org/jira/browse/CALCITE-1788
https://calcite.apache.org
@ApacheCalcite
@julianhyde

THE APACHE SOFTWARE FOUNDATION

Apache Calcite

HORTONWORKS