How to count really fast: A HyperLogLog story.

Marius Steinbakken Bråten, Reduan Azouaghe

Introduction

1. What is the problem?
    a. The count-distinct problem
2. How to solve the problem?
    a. Naive solutions
    b. HyperLogLog
3. How does HyperLogLog?
    a. Probabilistic intuition
    b. The algorithm

The count-distinct problem

The count-distinct problem is defined as following; “[...] the problem of finding the number of distinct elements in a data stream with repeated elements” - Wikipedia, Count-distinct problem

Translation: “In a bag containing coloured balls, how many unique balls do you have?”

Essentially: How do you count the colours really fast?

Possible solutions

Put them in a list.

```
unique_balls <- List
function add_ball(ball)
   if ball not in unique_balls # “in” is a linear scan.
       unique_balls.append(ball)
print(unique_balls.size())
```

But what if you had a gazillion balls?

- You have to check the whole list every time you append something. Slow!

Possible solutions (cont.)

Put them in a hash-structure.

```
unique_balls <- Hash based structure
function add_ball(ball)
   unique_balls.add(ball)
print(unique_balls.size())
```

But what if you had a gazillion balls?

- You have to create space for all possible colours before you even start. 48bit RGB space defines 281,474,976,710,656 colours. That’s about 1.5 petabytes (1500 terabytes). Expensive!

What to do?

HyperLogLog!

What is HyperLogLog?

“HyperLogLog is an algorithm for the count-distinct problem, approximating the number of distinct elements in a multiset” - Wikipedia, HyperLogLog

Translation: “estimate how many distinctly coloured balls we have in our bag”.

The HyperLogLog intuition

- Let’s say you throw one coin 3 times.
- What are the chances you get 3 tails in a row?
  - 1/2^3 = 1/8, which means you have to throw on average 8 times.
  - Pretty common, takes probably less than 3 minutes.

- Let’s say you throw one coin 30 times.
- What are the chances you get 30 tails in a row?
  - 1/2^30 = 1/1 073 741 824, which means you have to throw on average 1.073 billion times.
  - Uncommon. Would take a couple of lifetimes to do.

The HyperLogLog intuition (cont.)

- Essentially the length of your longest streak gives insights into how long you've been throwing coins.
- A streak of 3 would mean that you *probably* haven't thrown that manny coins.
- A streak of 30 would mean that you *probably* have thrown a lot of coins.

- HyperLogLog arms itself with this intuition to estimate how many many unique balls there are in your bag of balls

How does HyperLogLog work?

- Instead of using streaks in coin tosses, it uses streaks of zeros in binary

1. Hash the input into a binary value
2. Count the number of zeros at the end of the hash
3. Use the first few digits of the hash to determine the register
4. Save the count in the register
    - If it is higher than what is already in the register
5. Repeat

- There should be an equal chance to get 1’s and 0’s in the hashed value

How does HyperLogLog work? (cont.)

- We take a register with a counter
  - Let us say the value is 30
- The probability of seeing a streak of 30 zeros in a binary hash is 1/2^30 ≈ 1/1.073 billion
- Which means we have probably seen around 1.073 billion unique balls at this point
- This is used in the calculation of the estimate

Accuracy

- HyperLogLog uses harmonic mean to make an estimate
  - Less weight to large values, more weight to small values
  - Helps combat outliers skewing the estimate
- More registers leads to higher accuracy
  - But more memory usage
    - HyperLogLog is supposed to save memory
- There is a tradeoff between accuracy and memory usage

Cons with HyperLogLog

- It is probabilistic

- It never gives us the actual answer to our question “how many unique balls do we have”. It gives an estimation based on probability

- We *could* get a streak of 30 zeroes even though in reality we only have 5 balls. Highly unlikely, but still possible
  - However there are ways to to make a miss like this even more *unlikely*
  - The standard error is about 2%

Pros with HyperLogLog

- Fast!
  - We compare the hashed value to a register
  - We do not have to compare it to actual previous values
- Saves memory!
  - We avoid having to store previously seen values
- Adjustable!
  - You can keep multiple registers for increased accuracy
    - Costs more memory; accuracy is not free
- Scalable!
  - Multiple computers can count in parallel
  - Unite the registers

Thank you for your attention.

Sources:

- Count-distinct problem: https://en.wikipedia.org/wiki/Count-distinct_problem
- Color depth: https://en.wikipedia.org/wiki/Color_depth
- HyperLogLog: https://en.wikipedia.org/wiki/HyperLogLog
- HyperLogLog: https://chengweihu.com/hyperloglog/