# Mehrdad Pazooki

Founder of **TranQuant,** a data marketplace.

Founder of **Toronto Apache Spark** meetup group.

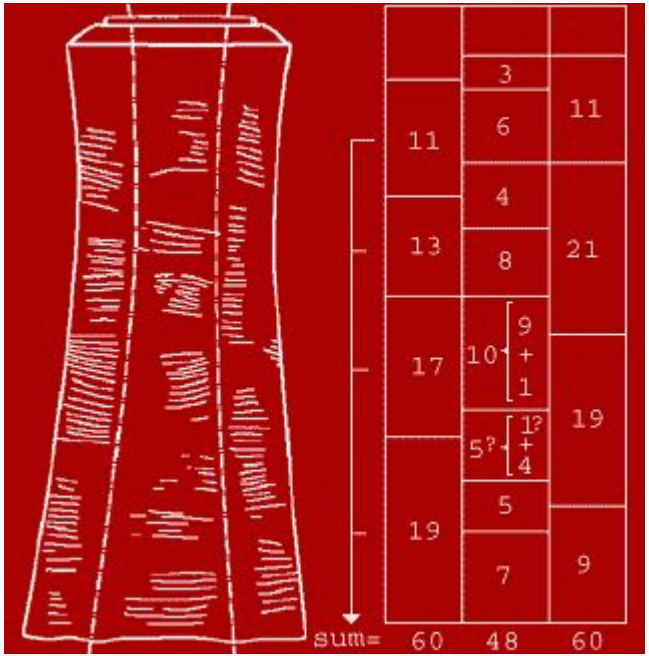
Email: mehrdad@tranquant.com

Slides: https://github.com/pazooki/presentations

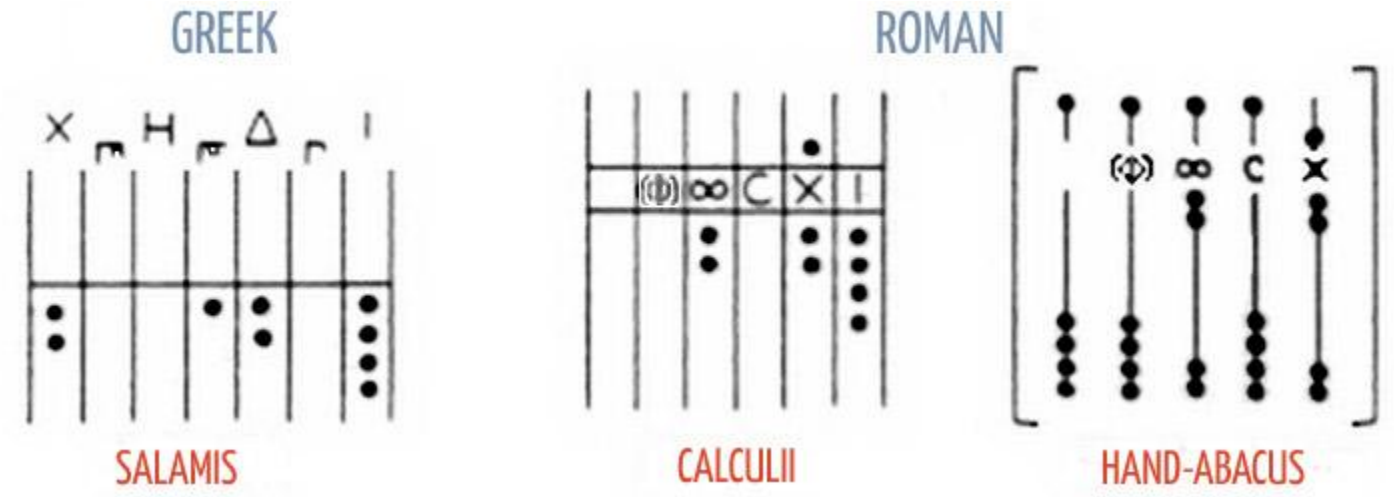
# Count-distinct Problem

Count-distinct problem in large datasets.

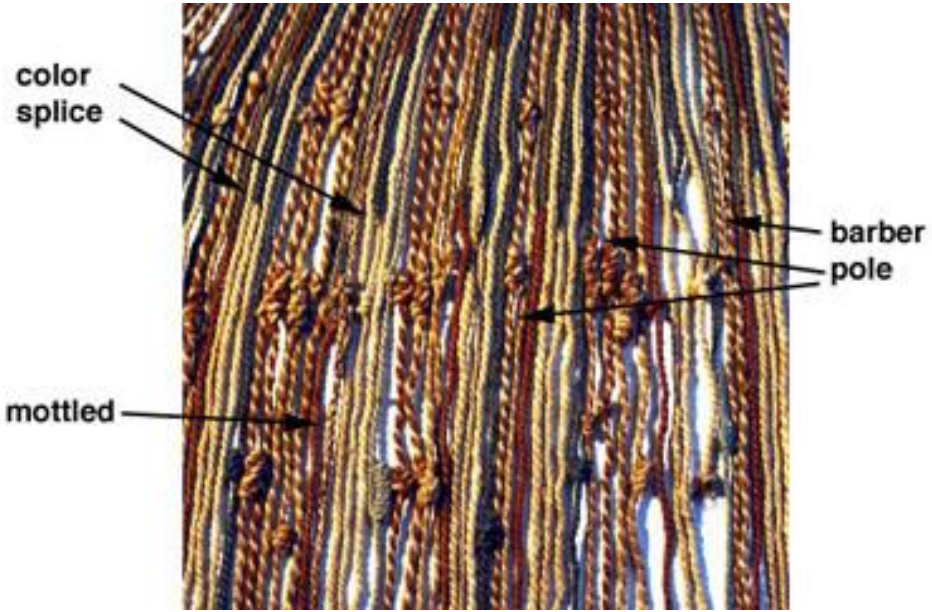
# Ishango Bone (20,000 Years Old!)

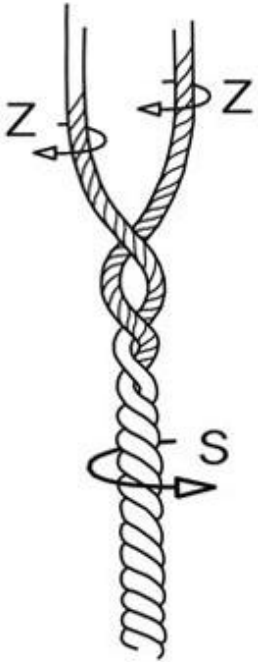


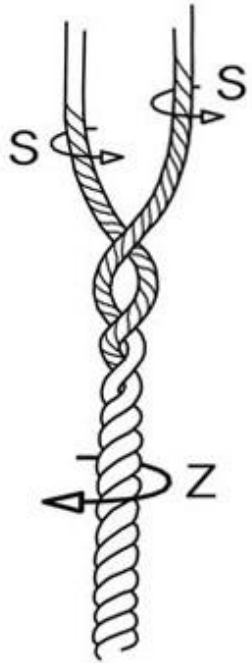
# Ancient Counting Devices (500 BCE - 500 CE)



# Khipu (1400)



# Soroban (1930)


SEIKO
UC-2100
RETURN

## Count-distinct problem

Finding the number of distinct elements in a dataset with repeated elements.

# Count-distinct Problem

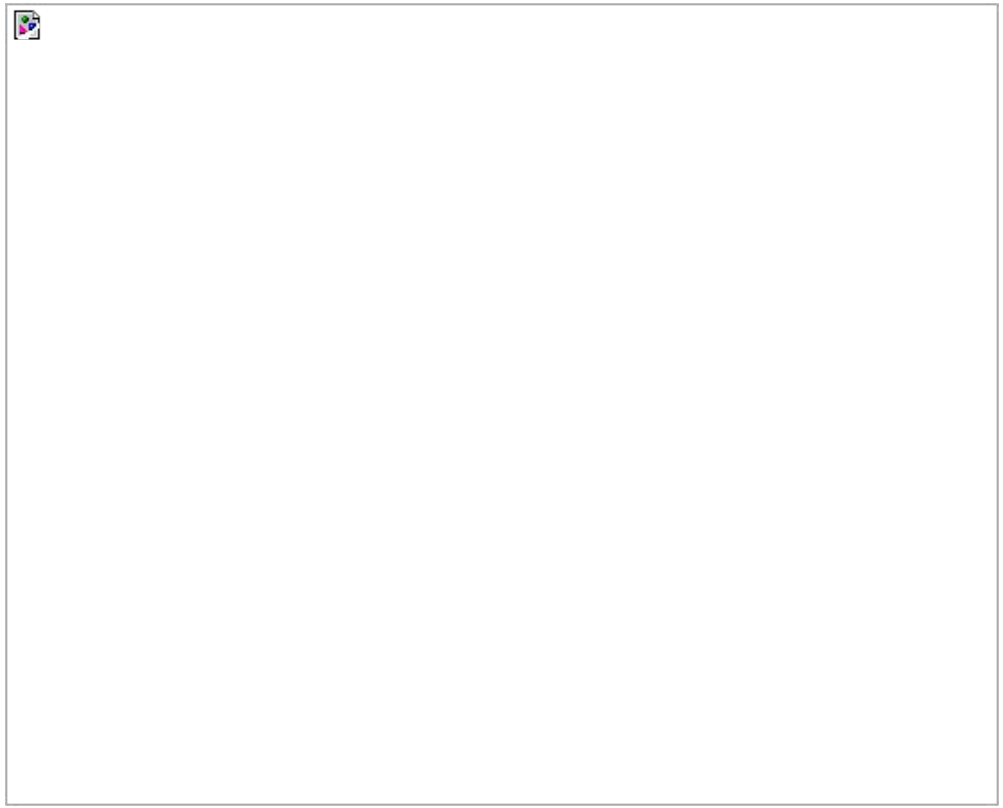
# or

# Cardinality Estimation Problem

# Number of unique users interested in “**blue shoes**”

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Event_ID | User_ID | Segment | Event_Type | | | | | | |
| 2 | 1 | 26 | red shoes | clicked | | Segment | Event Type | Total | Uniques | |
| 3 | 2 | 3 | blue shoes | clicked | | blue shoes | clicked | 20 | 5 | |
| 4 | 3 | 40 | green shoes | viewed | | blue shoes | viewed | 35 | 10 | |
| 5 | 4 | 38 | green shoes | clicked | | green shoes | clicked | 20 | 5 | |
| 6 | 5 | 4 | red shirts | viewed | | green shoes | viewed | 35 | 10 | |
| 7 | 6 | 5 | white bags | clicked | | red shoes | clicked | 20 | 5 | |
| 8 | 7 | 2 | red shirts | clicked | | red shoes | viewed | 35 | 10 | |
| 9 | 8 | 21 | red shoes | clicked | | white bags | clicked | 20 | 5 | |
| 10 | 9 | 23 | blue shoes | viewed | | white bags | viewed | 35 | 10 | |
| 11 | 10 | 26 | red shoes | clicked | | yellow shirts | clicked | 20 | 5 | |
| 12 | 11 | 13 | yellow shirts | clicked | | yellow shirts | viewed | 35 | 10 | |
| 13 | 12 | 24 | green shoes | viewed | | red shirts | clicked | 20 | 5 | |
| 14 | 13 | 3 | green shoes | viewed | | red shirts | viewed | 35 | 10 | |
| 15 | 14 | 16 | red shoes | clicked | | | | | | |
| 16 | 15 | 5 | red shirts | viewed | | | | | | |

# Distributed Systems



# Probabilistic Data Structures

# Vs

# Deterministic Data Structures

# Deterministic Data Structures

The result of following functions:

- Insert
- Find
- Delete
- Count
- ...

Are always consistent.

Example: Set, Arrays,...

# Probabilistic Data Structures

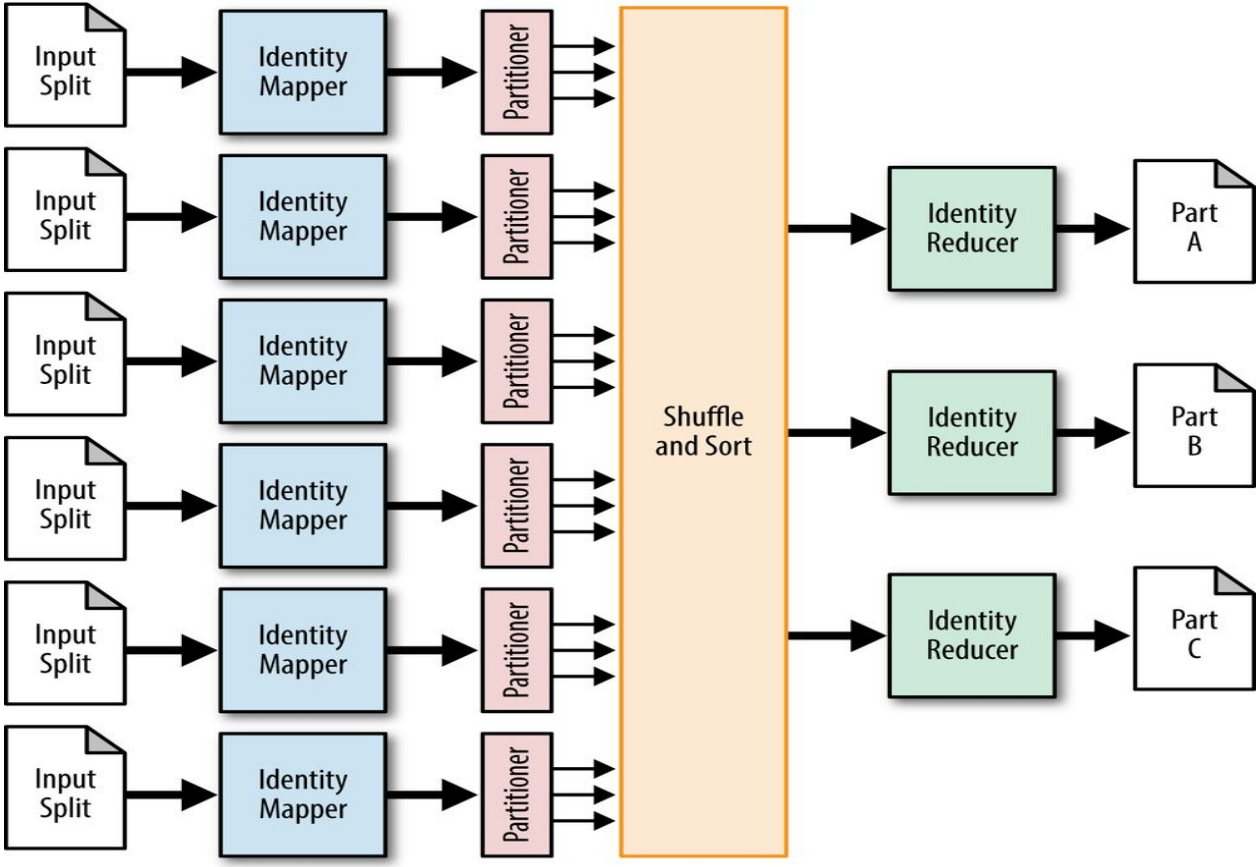
The result of following functions:

- Insert
- Find
- Delete
- Count
- ...

Are not always consistent.

Examples: BloomFilter, HyperLogLog, Count-min Sketch, MinHash,...

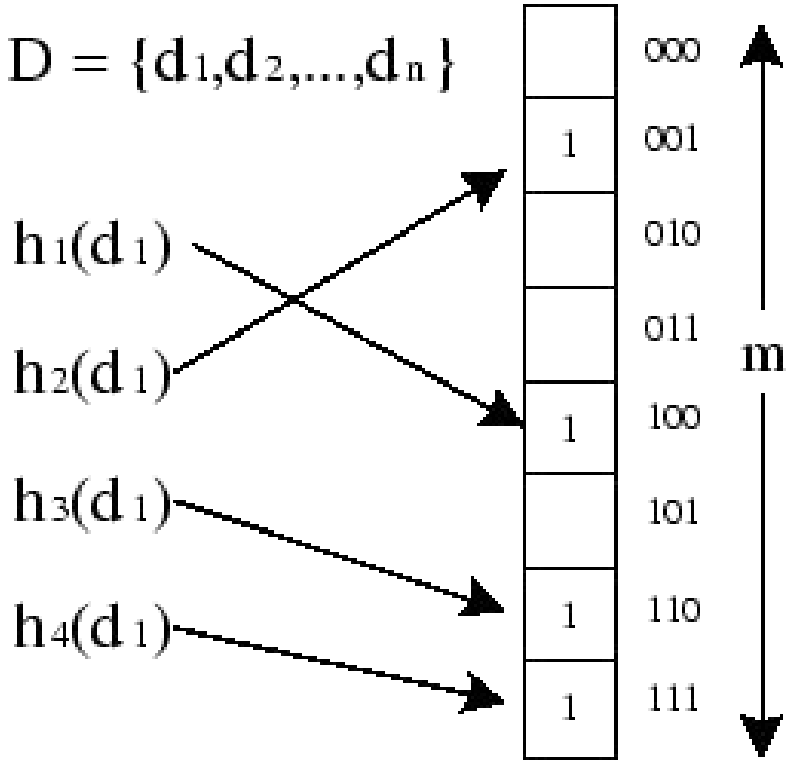
# Map/Reduce


Input Split
Identity Mapper
Partitioner
Input Split
Identity Mapper
Partitioner
Input Split
Identity Mapper
Partitioner
Input Split
Identity Mapper
Partitioner
Input Split
Identity Mapper
Partitioner
Input Split
Identity Mapper
Partitioner
Shuffle and Sort
Identity Reducer
Part A
Identity Reducer
Part B
Identity Reducer
Part C

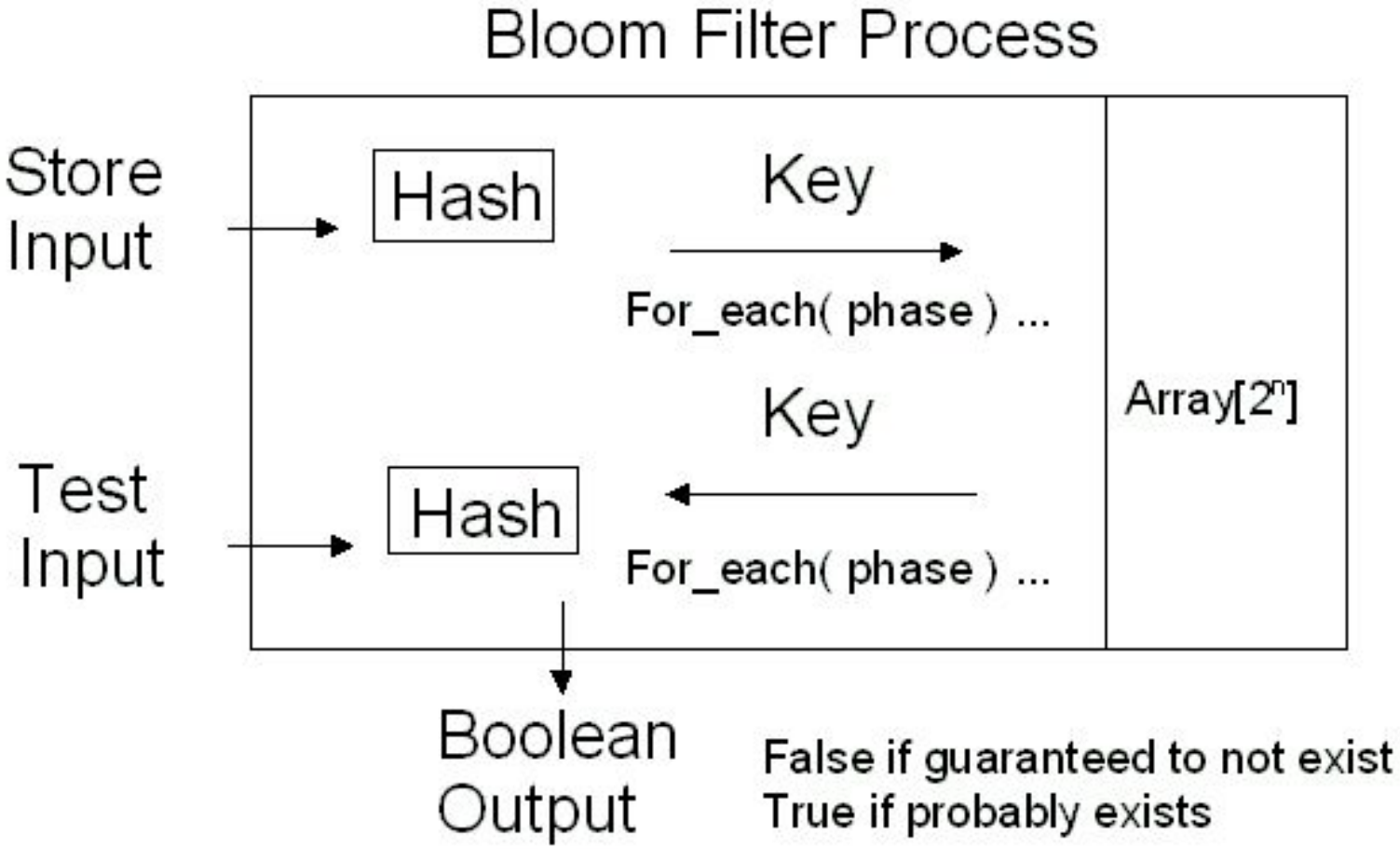
# Number of unique users interested in “**blue shoes**”

| | A | B | C | D | E | F | G | H | I | J |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Event_ID | User_ID | Segment | Event_Type | | | | | | |
| 2 | 1 | 26 | red shoes | clicked | | Segment | Event Type | Total | Uniques | |
| 3 | 2 | 3 | blue shoes | clicked | | blue shoes | clicked | 20 | 5 | |
| 4 | 3 | 40 | green shoes | viewed | | blue shoes | viewed | 35 | 10 | |
| 5 | 4 | 38 | green shoes | clicked | | green shoes | clicked | 20 | 5 | |
| 6 | 5 | 4 | red shirts | viewed | | green shoes | viewed | 35 | 10 | |
| 7 | 6 | 5 | white bags | clicked | | red shoes | clicked | 20 | 5 | |
| 8 | 7 | 2 | red shirts | clicked | | red shoes | viewed | 35 | 10 | |
| 9 | 8 | 21 | red shoes | clicked | | white bags | clicked | 20 | 5 | |
| 10 | 9 | 23 | blue shoes | viewed | | white bags | viewed | 35 | 10 | |
| 11 | 10 | 26 | red shoes | clicked | | yellow shirts | clicked | 20 | 5 | |
| 12 | 11 | 13 | yellow shirts | clicked | | yellow shirts | viewed | 35 | 10 | |
| 13 | 12 | 24 | green shoes | viewed | | red shirts | clicked | 20 | 5 | |
| 14 | 13 | 3 | green shoes | viewed | | red shirts | viewed | 35 | 10 | |
| 15 | 14 | 16 | red shoes | clicked | | | | | | |
| 16 | 15 | 5 | red shirts | viewed | | | | | | |

# BloomFilter




A Bloom Filter that uses 4 hash functions and has a size of m=8 bits.


Bloom Filter Process
Store Input
Hash
Key
For_each( phase ) ...
Key
Array[2^n]
Test Input
Hash
For_each( phase ) ...
Boolean Output
False if guaranteed to not exist
True if probably exists

# BloomFilter

- Compact with controllable error rate
- Great for fast filtering (is value V in set S?)
  - Fraud Detection
  - DDOS Attack Prevention
  - …
- Great for problems not sensitive to false positive results
- Not that great for distinct count of large datasets
- It will be sparse for small datasets
- Dynamic Block-Partitioned BloomFilters

# HyperLogLog

Q: How many distinct elements are in an infinite stream of data?

Similar to

Q: How many times the coin is flipped in coin flipping?

# Coin Flipping

- Long runs of **heads** in random series are rare
- The longer you look, the more likely you will find one
- Long runs are very rare and are correlated with how many coins you've flipped

# Basic HyperLogLog Algorithm

n = 0

For each input item:

    Hash item into bit string

    Count trailing zeros in bit string

    if this count > n:

        Let n = count

Estimated Cardinality (“Count Distinct”) = 2^n

# HyperLogLog

- Great for distinct count of large datasets
  - Fixed Memory
- Some set operations:
  - Union
- Higher error ratio
- Limited spectrum for set Intersection operation
- The HyperLogLog algorithm is able to estimate cardinalities of 10^9 with a typical error rate of 2%, using 1.5 kB of memory.

# Approximate Count in Apache Spark

**countApproxDistinct**(*relativeSD=0.05*)

> **Note:** Experimental

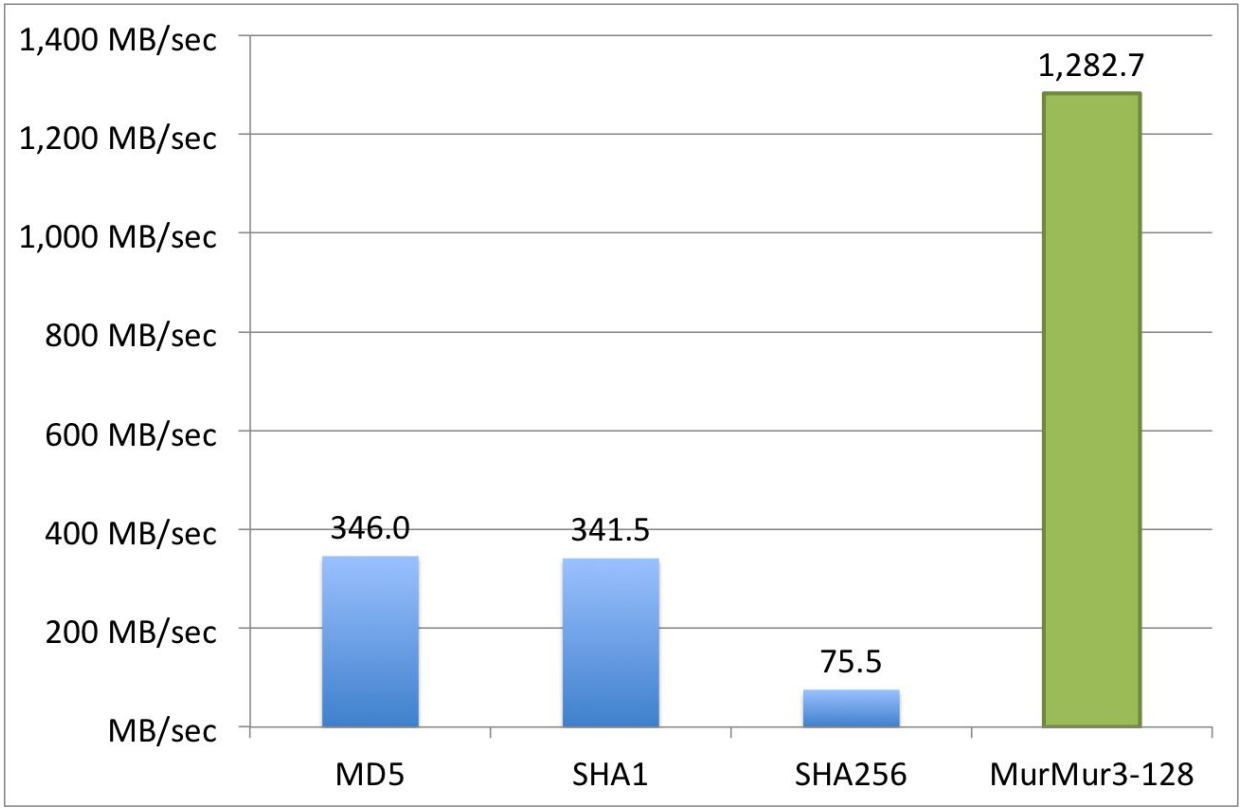
Return approximate number of distinct elements in the RDD.

The algorithm used is based on streamlib's implementation of "HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm", available here.

**Parameters:** **relativeSD** – Relative accuracy. Smaller values create counters that require more space. It must be greater than 0.000017.

```
>>> n = sc.parallelize(range(1000)).map(str).countApproxDistinct()
>>> 900 < n < 1100
True
>>> n = sc.parallelize([i % 20 for i in range(1000)]).countApproxDistinct()
>>> 16 < n < 24
True
```

# Hash Functions



# References


- https://highlyscalable.wordpress.com/2012/05/01/probabilistic-structures-web-analytics-data-mining/
- https://en.wikipedia.org/wiki/Count-distinct_problem
- https://en.wikipedia.org/wiki/Category:Probabilistic_data_structures
- http://www.slideshare.net/a235/probabilistic-data-structures-and-approximate-solutions
- http://dl.acm.org/citation.cfm?doid=2452376.2452456
- http://www.ee.ryerson.ca/~elf/abacus/images/fig-antiquity.JPG
- https://pdfs.semanticscholar.org/5da8/bf81712187712aed159aed62e38fb012872e.pdf

# Thank You!

Mehrdad Pazooki

Founder of **TranQuant,** a data marketplace.

Founder of **Toronto Apache Spark** meetup group.

Email: mehrdad@tranquant.com

Slides: https://github.com/pazooki/presentations