Putting it all together

Systems design

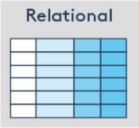
In this lecture

1. Quick recap of scale

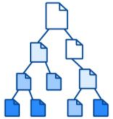
2. A full systems example putting things together
   - ▷ How to compute data sizes?
   - ▷ Computing a large join?
   - ▷ Building an index

Data models

Structured
(e.g. ads, purchases, product tables)
[aka relational tables]



Semi-structured
(e.g. user profile, web site activity)
[aka JSON documents]



irst_name	last_name	cell	city	year_of_birth	location_x	location_y
'Mary'	'Jones'	'516-555-2048'	'Long Island'	1986	'-73.9876'	'40.7574'

ID	user_id	profession
10	1	'Developer'
11	1	'Engineer'

ID	user_id	name	version
20	1	'MyApp'	1.0.4
21	1	'DocFinder'	2.5.7

ID	user_id	make	year
30	1	'Bentley'	1973
31	1	'Rolls Royce'	1965

```
last_name: "Jones",
cell: "516-555-2048",
city: "Long Island",
year_of_birth: 1986,
location: {
        type: "Point",
        coordinates: [-73.9876, 40.7574
},
profession: ["Developer", "Engineer"],
apps: [
  { name: "MyApp",
    version: 1.0.4 },
  { name: "DocFinder",
    version: 2.5.7 }
 ],
cars: [
   { make: "Bentley",
     year: 1973 },
   { make: "Rolls Royce",
     year: 1965 }
 ]
```

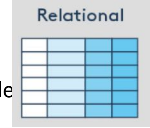
Hybrid Data Systems

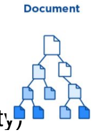
In reality, a mix of systems -- e.g., Amazon/e-commerce site

Data

Structured

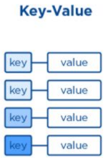
(e.g. ads, purchases, product table)





Semi-structured

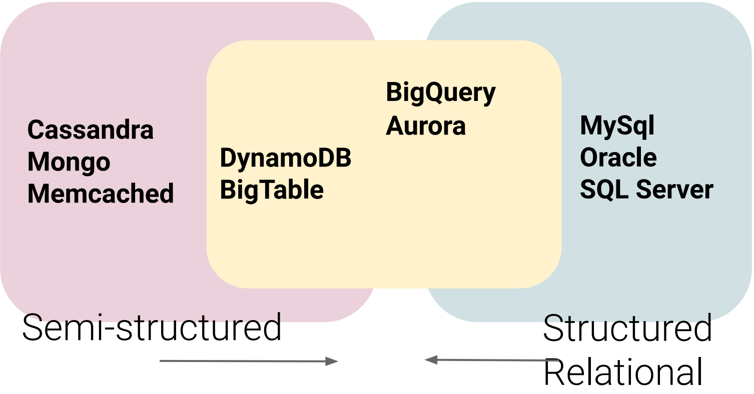
(e.g. user profile, web site activity)



Unstructured

(e.g. images, videos, docs)

DB Service



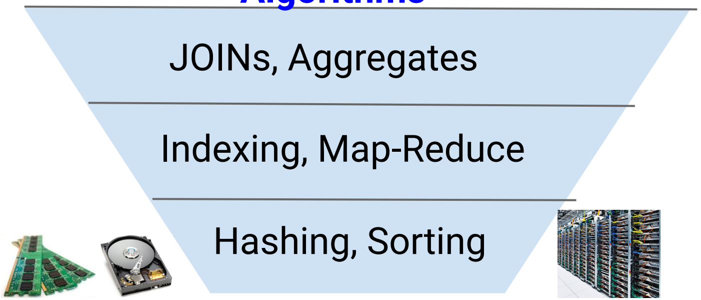
Language/Tools

Hybrid future

- Built on same Lego blocks
- Past: SQL → noSQL → newSQL
- Now: hybrid SQL + pandas/ML

Example: [Alibaba's data analysis design]

Algorithms



Key Starter Questions for:

Scale, Scale, Scale

1. How to scale to large data sets?
   - Is data relational, or unstructured or ...?
   - Is data in Row or Column store?
   - Is data sorted or not?

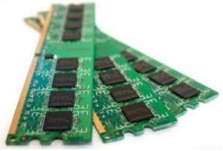
2. How do we organize search values?
   - E.g., Hash indices, B+ trees

3. How to JOIN multiple datasets?
   - E.g., SortMerge, HashJoins

Big Scale Lego Blocks

Roadmap

Primary data structures/algorithms



	Hashing	Sorting
RAM	HashTables ($hash_i$(key) --> location)	BucketSort, QuickSort MergeSort
Disk	HashFunctions ($hash_i$(key) --> location)	MergeSortedFiles MergeSort
Cluster	HashFunctions ($hash_i$(key) --> location)	MergeSort

IO analysis

Recap

Sorting of relational T with N pages

$$\sim 2N\left(\left\lceil \log_B \frac{N}{2(B+1)} \right\rceil + 1\right)$$

Sort N pages with B+1 buffer size

(vs n log n, for n tuples in RAM. Negligible for large data, vs IO -- much, much slower)

$\sim 2N$

Sort N pages when N ~= B

(because (Log_B 0.5) < 0)

$\sim 4N$

Sort N pages when N ~= 2*B^2

(because (Log_B B) = 1)

SortMerge and HashJoin for R & S

~3 * (P(R) + P(S)) + OUT

Where P(R) and P(S) are number of pages in R and S, when you have enuf RAM

~1 * (P(R) + P(S)) + OUT

For SortMerge, if already sorted

For HashJoin, if already partitioned

In this lecture

1. Quick recap of scale

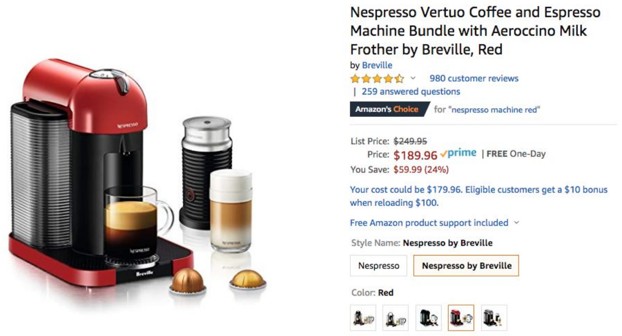
2. A full systems example putting things together
   - ▹ How to compute data sizes?
   - ▹ Execute a large join?
   - ▹ Build an index

Systems Design Example:

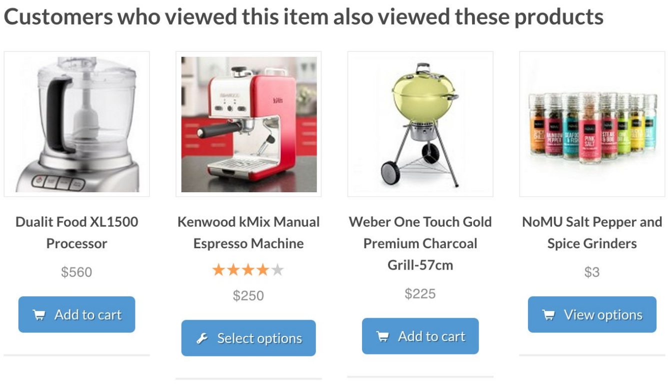
Product Search & CoOccur

Billion products

User searches for “coffee machine”



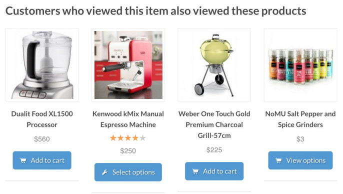
Product recommendations



Systems Design Example:

Product Search & CoOccur

Counting popular product-pairs



Story: Amazon/Walmart/Alibaba (AWA) want to sell products

1. AWA wants fast user searches for product
2. AWA shows 'related products' for all products so users can explore
   - Using collaborative filtering ('wisdom of crowds') from historical website logs.
   - Each time a user views a set of products, those products are related (co-occur)

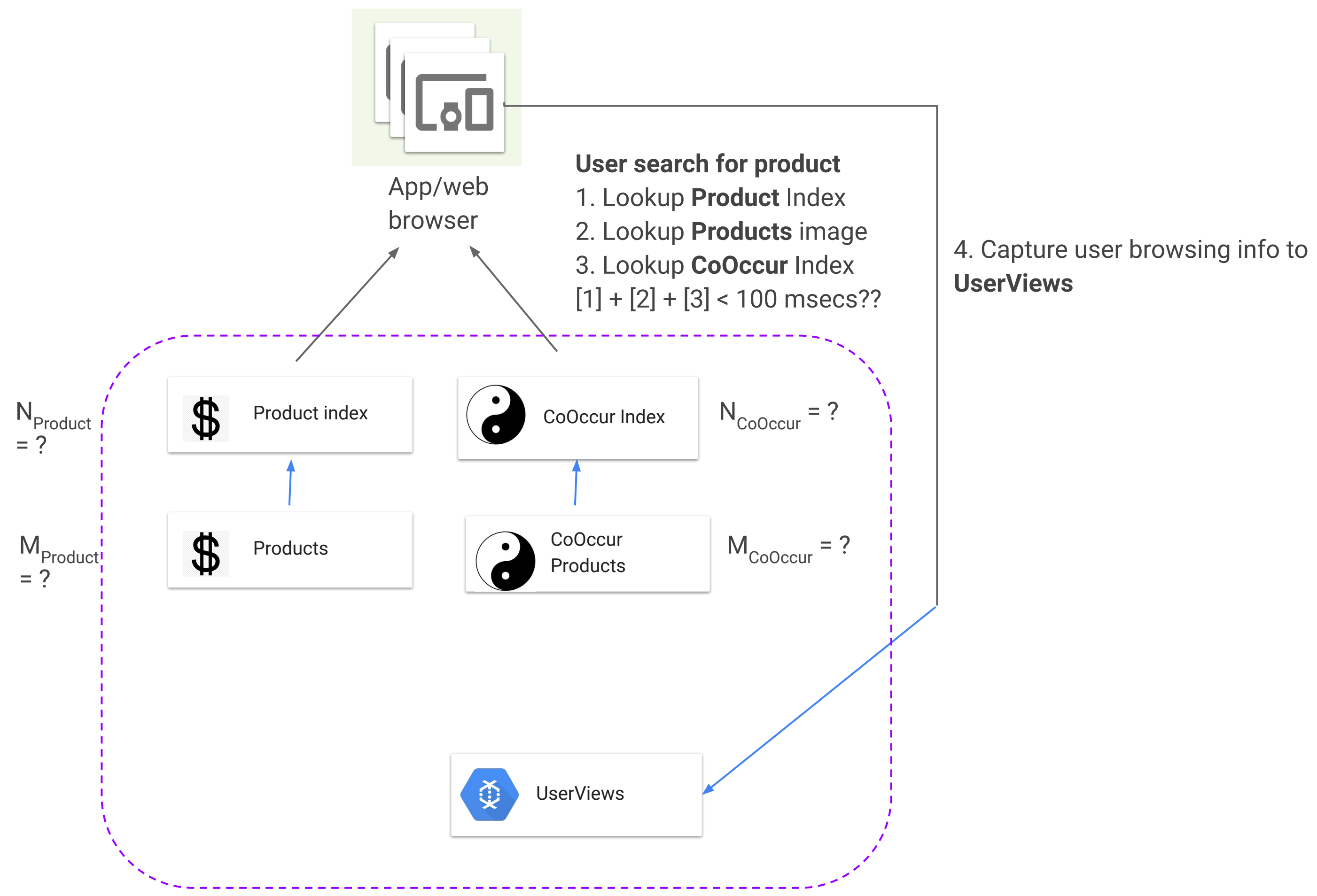
⇒ Goal: compute product pairs and their co-occur count, across all users

Data input:

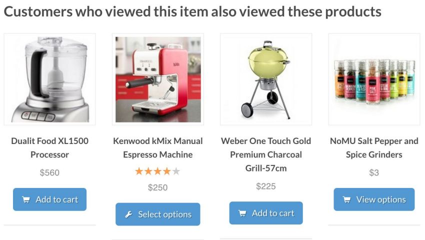
- AWA has 1 billion products. Each product record is ~1MB (descriptions, images, etc.).
- AWA has 10 billion UserViews each week, from 1 billion users. Stored in UserViews, each row has <userID, productID, viewID, viewTime>.

Data Systems Design Example:

Product Search& CoOccur



Counting in RAM
(pre-CS145)



Counting product views for billion product PAIRs

UserViews(UserId, ProductID, …, ...)

	Nespresso Coffee
	Bread maker
	Kenwood Espresso
	...

CoOccur(ProductID1, ProductID2, count)

Nespresso Coffee	Bread Maker	301
Bread maker	Kenwood Espresso	24597
Kenwood Espresso	Bike	22
		...

Algorithm: For each user, product p_i p_j
CoOccur[p_i, p_j] += 1

Pre-cs145

Compute in RAM

(Engg approximations)

Counting product views

Input size (4 bytes for user, 4 bytes for productid)	~1Bil * [4 + 4] = 8 GB
Output size (4 bytes for productid, 4 bytes for count)	~1Bil * [4 + 4] = 8 GB

Trivial

Counting product pair views

'Trivial?'
(if you have ~25 Billion$, at 100$/16GB RAM)

Plan #1: P * P matrix for counters in RAM (4 bytes for count)

- RAM size = 1 Billion * 1Billion * 4 = 4 Million TBs
- [*Note: We'll keep counter(p_i , p_j) and counter(p_j , p_i) in these examples. Why? We want to lookup cooccur counters both ways*]

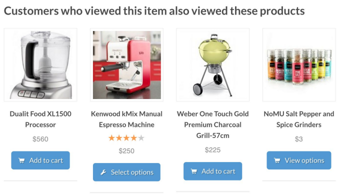
Plan #1 (on disk): Let OS page into memory as needed

- Worst case #1 > 100 million years
  (if you seek to update each counter)

Systems Design Example:

Product CoOccur

Counting popular product-pairs

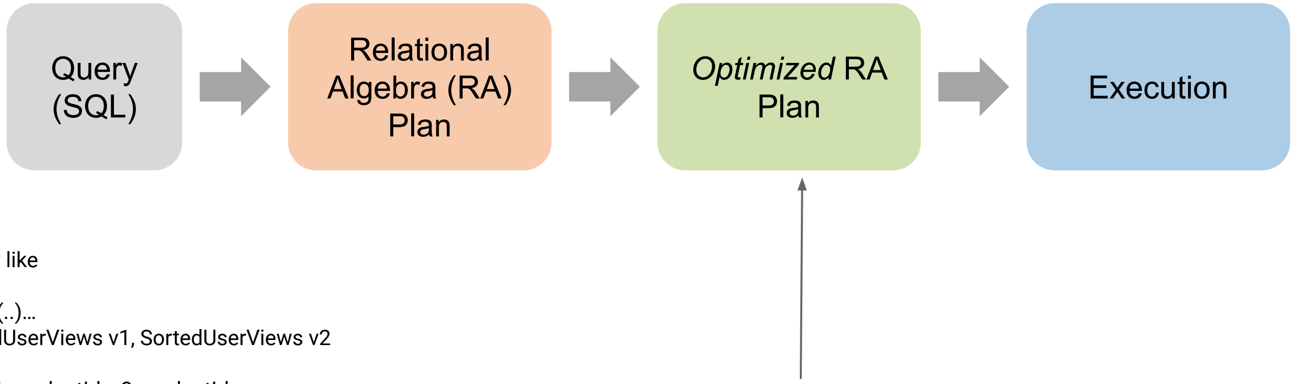


Your mission: Design an efficient system to compute co-occur counts on *Sundays* from weekly logs and produce a CoOccurCount table <productID, productID, count>

1. AWA's data quality magicians recommend
   - (a) keep only top billion popular pairs, and (b) drop pairs with co-occur counts less than million.
   - (c) Also, assume users view ten products on average each week (*User is interested in ~10 products/week, not 1000s*).

2. For simplicity, SortedUserViews is stored sorted by <userID, productID>.
   - You can sequentially scan the log and produce co-occurring product pairs for each user. In other words, output (p_i, p_j) if a user viewed products p_i and p_j.
   - This "stream" of tuples (TempCoOccur) may then be (a) stored on disk or (b) discarded after updating any data structures.

Post CS 145



Write a query like

SELECT TOP(..)…
FROM SortedUserViews v1, SortedUserViews v2
WHERE ...
GROUP BY v1.productId, v2.productId
HAVING count(*) > 1 million

Optimize,

Evaluate design plans

Build Query Plans

Analyze Plans

1. For SFW, Joins queries
    a. Sort? Hash? Count? Brute-force?
    b. Pre-build an index? B+ tree, Hash?

2. What statistics can I keep to optimize?
    a. E.g. Selectivity of columns, values

Cost in I/O, resources?
To query, maintain?

Systems Design Example:

Product CoOccur

Plans?

~~Plan#1: With 1 machine, use RAM to count (Cost = 25B$ or > 100 milllion years).~~

Plan#2: With 1 machine

Plan 2

1. Scan SortedUserViews. For each user, **append** <p_i, p_j> to a file TempCoOccurLog if the user has viewed p_i and p_j. (i.e., produce per-user co-occur product pair. *Append to log ⇒ No seek...*)
2. Externally sort TempCoOccurLog on disk, so identical product pairs are adjacent to each other in the sorted file
3. Scan sorted TempCoOccurLog. With a single pass, you can count co-occur pairs. Drop co-occur pairs with < 1 million.

Nespresso	Iphone
...	
...	
Nespresso	Iphone
...	
...	
...	
Nespresso	Iphone

TempCoOccurLog
(After Step 1)

.....	
Nespresso	Iphone
Nespresso	Iphone
Nespresso	Iphone
.....	

Sorted TempCoOccurLog
(After Step 2)

Nespresso	Iphone	3

Count sorted TempCoOccurLog
(After Step 3)

Systems Design Example: Product CoOccur

Pre-design

	Size	Why?
ProductId	4 bytes	1 Billion products ⇒ Need at least 30 bits (2^30 ~= 1 Billion) to represent each product uniquely. So use 4 bytes.
UserID	4 bytes	"
ViewID	8 bytes	10 Billion product views.
Product	1 PB	1 Billion products of 1 MB each
SortedUserViews	240 GB (4 M pages)	Each record is <userID, productID, viewID, viewtime>. Assume: we use 8 bytes for viewTime. So that's 24 bytes per record. 10 Billion*24 bytes = 240 GBs.
CoOccur (for top 1 Billion)	12 GB	The output should be <productID, productID, count> for the co-occur counts. That is, 12 bytes per record (4 + 4 + 4 for the two productIDs and 4 bytes for count). To keep top billion product pairs (as recommended by AWA data quality), you need 1 billion * 12 bytes = 12 GBs.
TempCoOccurLog (assume: ~10 product views/user)	800 GB (12.5 M pages)	# product pairs produced: 1 billion users * 10^2 = 100 billion Size @8 bytes/record (productID, productID) = 800 GBs

Systems Design Example:

Product CoOccur

Plan #2

Steps	Cost (time)	Why?
Scan SortedUserViews Append <p_i, p_j> to TempCoOccurLog		
Externally sort TempCoOccurLog on disk (Assume sort cost is ~2N, where N is number of pages for table and B is number of buffers, and B ~~ N)		
Scan TempCoOccurLog (sorted) and keep counts in CoOccur		

Systems Design Example:

Product CoOccur

Plan #2

Steps	Cost (IO)	Why?
Scan SortedUserViews Append <p_i, p_j> to TempCoOccurLog	4 M 12.5M	240GB (4 M pages) 800 GB (12.5M pages)
Externally sort TempCoOccurLog on disk (Assume sort cost is ~2N, where N is number of pages for table and B is number of buffers, and B ~~ N)	25M	IO cost is (appx) = 2*N = 2*12.5M
Scan TempCoOccurLog (sorted) and keep counts in CoOccur	12.5M	800 GB

Total IO cost = (4M+ 12.5M +25M + 12.5M) = 54M
Recall: Scan at 100 MBps, then time (secs) [assume, files are stored sequentially]
= (54M * 64 KB) /100 MBps = ~34.5K secs

Optimize,

Evaluate design plan 1, plan2, plan 3, ...

Build Query Plans

Analyze Plans

1. For SFW, Joins queries
   a. Sort? Hash? Count? Brute-force?
   b. Pre-build an index? B+ tree, Hash?

2. What statistics can I keep to optimize?
   a. E.g. Selectivity of columns, values

Cost in I/O, resources?
To query, maintain?

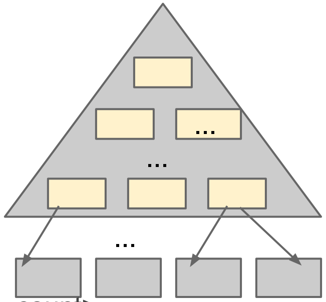
Systems Design Example:

Product CoOccur

B+ tree index

Build indexes with search key=productId. (Assume: CoOccur data may not be clustered)

CoOccur Index
<pid, pointer>



CoOccur
<pid, cooccur_pid, count>

$N_{CoOccur}$: 167K pages (i.e. 1 billion rows * (4 + 8) bytes each in 64KB pages)

$M_{CoOccur}$: 1 billion rows * (4 + 4+ 4) bytes each (~167k pages)

For Index on Product data? [Recall: 1 billion tuples * 1 M B each = 1 PB]

- $M_{Product}$: 1 PB/64 KB = 15.6 Billion pages
- $N_{Product}$: <pid, pointer> = 167K pages

Systems Design Example:

Product CoOccur

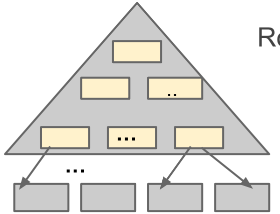
B+ tree index

N	167,000	From previous page
How large is f?	~5460	4 bytes for productId + 8 bytes for pointers @ 64KB/page f * 4 + f*8 <= 64k ⇒ f ~= 5460

Recall We need a B+ tree of height h = $\lceil \log_f N \rceil$

CoOccur Index
<pid, pointer>

CoOccur
<pid, cooccur_pid, count>



Root: 1 page, with 5460 pointers

Level 1: <= 5460 pages, with 5460 ptrs each to next level

Leaf: N = 167,000 pages

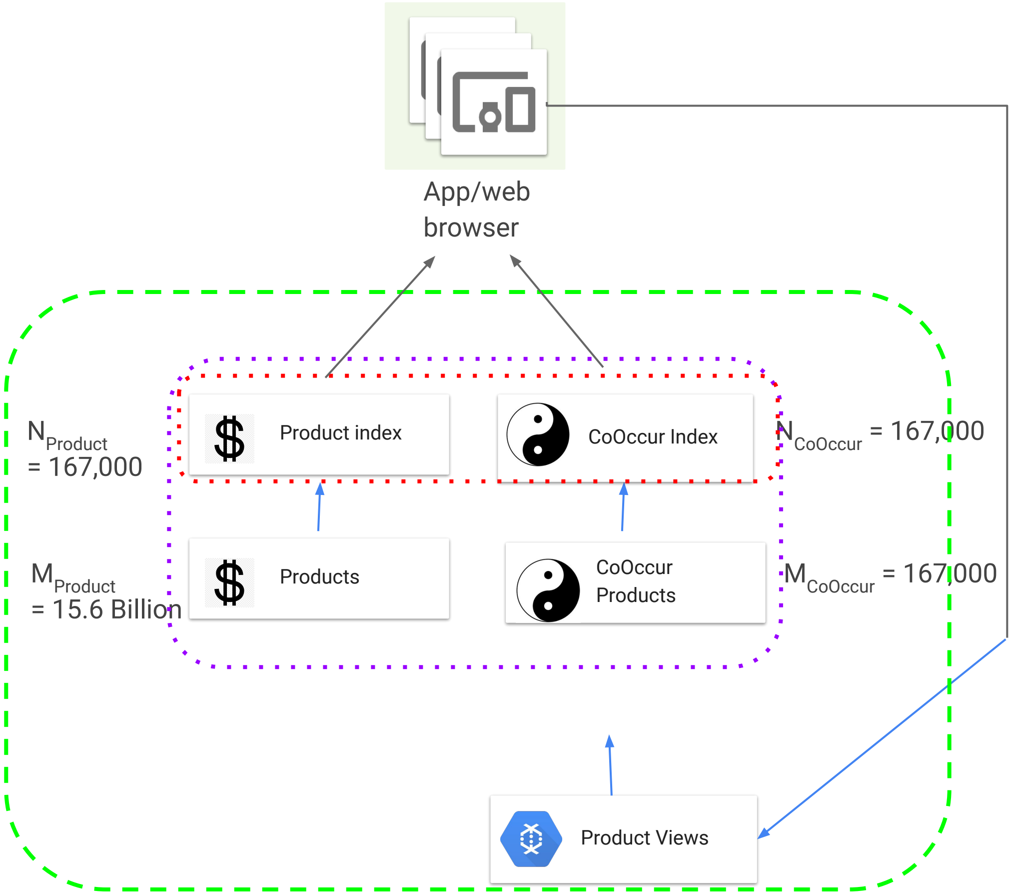
(h = 2, Each leaf can point upto 5460 search keys;

Note: can grow upto 5460^2 before needing h=3)

Data: 1 billion records with 12 bytes each

Data Systems Design Example:

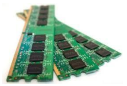
Product CoOccur



User latency?

1. Lookup Product Index
2. Lookup Products image
3. Lookup CoOccur Index

[1] + [2] + [3] < 100 msecs



~0.33M index Pages ~= 21 GBs
⇒ Keep in RAM

In RAM

Fast access

In datacenter

Data Systems Design Example:

Bigger Product CoOccur

Problem so far

- AWA's product catalog is 1 billion items. AWA has 10 billion product views each week, from 1 billion users. Each log record stores <userID, productID, viewID, viewtime>

Consider 1000x Bigger problem!

- Product catalog is 1 trillion items. AWA has 10 billion product views. Rest stays same

⇒ What changes?

Systems Design Example:

Product CoOccur

Pre-design

	Size	Why?
ProductId	8 bytes (vs 4bytes)	1 trillion products ⇒ Need at least 40 bits (2^40 ~= 1 Trillion) to represent each product uniquely. So use 8 bytes (i.e 64 bits).
UserID	4 bytes	“
UserViewsID	8 bytes	10 Billion product views.
Product	1000 PB	1 Trillion products of 1 MB each
Users	Unknown	
SortedUserViews	280 GB (vs 240 GB)	~~Each record is <userID, productID, viewID, viewtime>.~~ Assume: we use 8 bytes for viewTime. So that’s 28 bytes per record. 10 Billion*28 bytes = 280 GBs.
CoOccur	20 GBs (vs 12 GB)	The output should be <productID, productID, count> for the co-occur counts. That is, 20 bytes per record (8 + 8 + 4 for the two productIDs and 4 bytes for count). To keep top billion product pairs (as recommended by AWA data quality), you need 1 billion * 20 bytes = 20 GBs.
TempCoOccur (with UserSession assumption, of ~10 views/user)	1600 GB (vs 800 GB)	# product pairs produced: 1 billion users * 10^2 = 100 billion Size @16 bytes/record = 1600 GBs.

1000x larger catalog? < 2x increase in run time!

Data Systems Design

Popular Systems design pattern

1. Efficiently compute ‘batch’ of data (sort, hash, count)
2. Build Lookup index on result (b+ tree, hash table)
3. For ‘streaming’ data, update with ‘micro batches’

Popular problems

1. Related videos (youtube), people (Facebook), pages (web)
2. Security threats, malware (security), correlation analysis

Key Goal:

Scale, Scale, Scale

1. How to scale to large data sets?
   - Is data in Row or Column store?
   - Is data sorted or not?

2. How do we organize search values?
   - E.g., Hash indices, B+ trees

3. How to JOIN multiple datasets?
   - E.g., SortMerge, HashJoins

Histograms & IO Cost Estimation

Optimization Roadmap

Build Query Plans

Analyze Plans

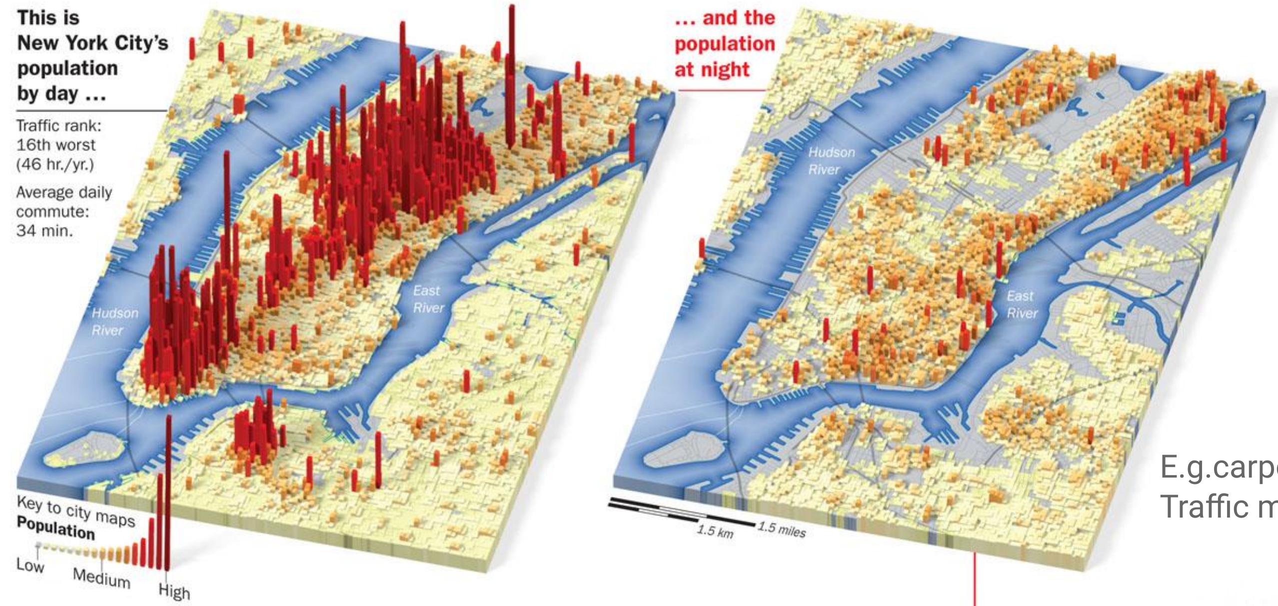
1. For SFW, Joins queries
   a. Brute-force? Sort? Hash? Count?
   b. Pre-build an index? B+ tree, Hash?

2. What statistics can I keep to optimize?
   a. E.g. Selectivity of columns, values

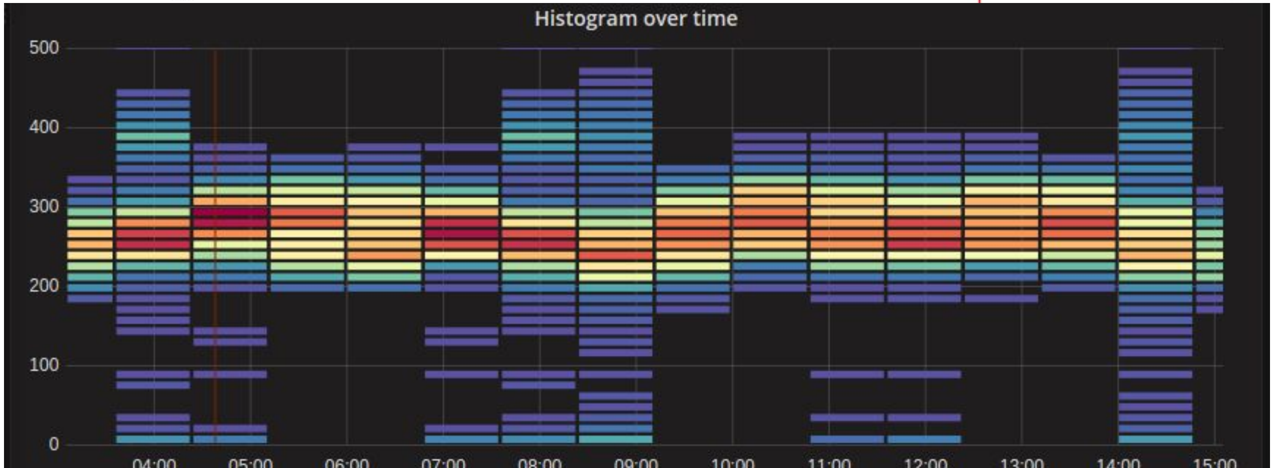
Cost in I/O, resources?
To query, maintain?

Example

Stats for spatial and temporal data



E.g.carpool or Traffic model

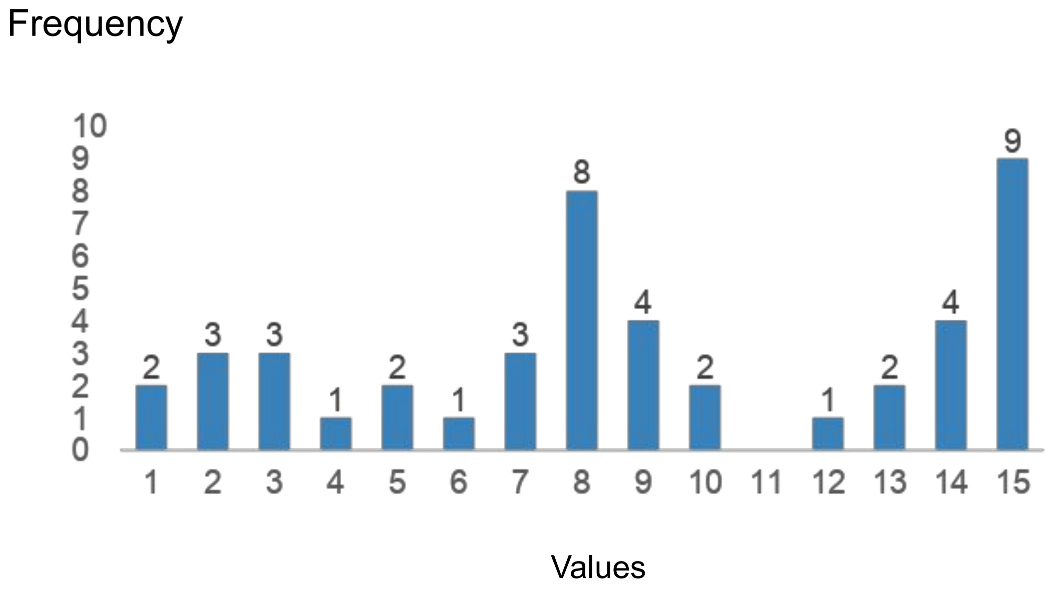


E.g. user traffic

Histograms

- A histogram is a set of value ranges ("buckets") and the frequencies of values in those buckets
- How to choose the buckets?
  - Equi-width & Equi-depth
- High-frequency values are very important(e.g, related products)

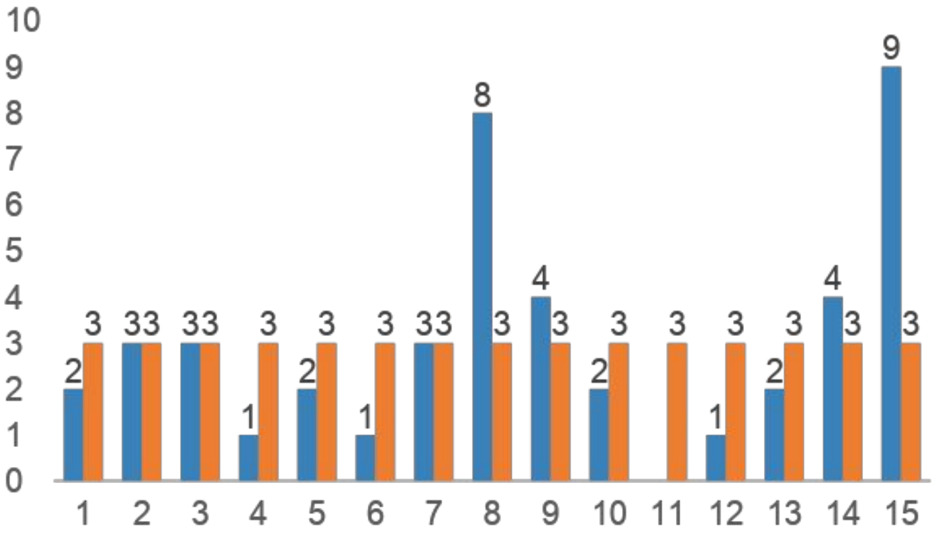
Example



How do we compute how many values between 8 and 10? (Yes, it's obvious)

Problem: counts take up too much space!

What if we kept average only?



E.g., uniform count = 3 (average)

How much space do the full counts (bucket_size=1) take?

How much space do the uniform counts (bucket_size=ALL) take?

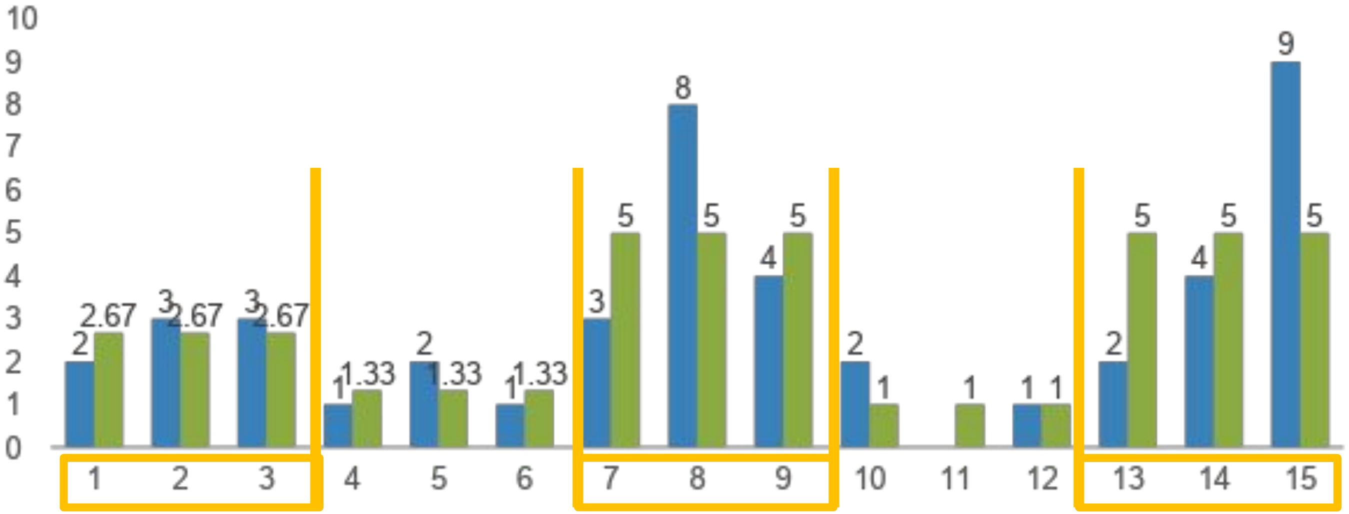
And Average Error?

Fundamental Tradeoffs

- Want high resolution (like the full counts)
- Want low space (like uniform)
- Histograms are a compromise!

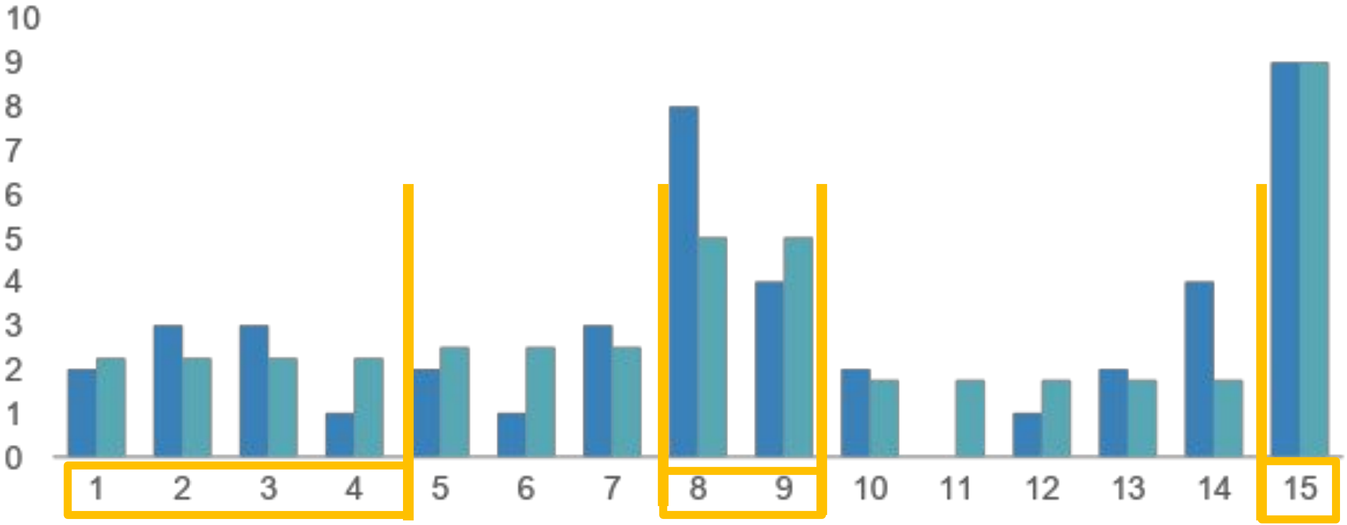
So how do we compute the “bucket” sizes?

Equi-width



Partition buckets into roughly same width (value range)

Equi-depth



Partition buckets for roughly same number of items (total frequency)

Histograms

- Simple, intuitive and popular
- Parameters: # of buckets and type
- Can extend to many attributes (multidimensional)

Maintaining Histograms

- Histograms require that we update them!
  - Typically, you must run/schedule a command to update statistics on the database
  - Out of date histograms can be terrible!

- Research on self-tuning histograms and the use of query feedback

Compressed Histograms

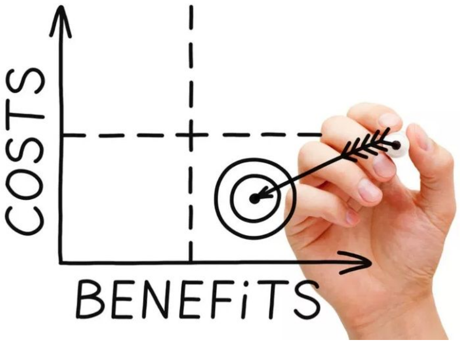
One popular approach

1. Store the most frequent values and their counts explicitly
2. Keep an equiwidth or equidepth one for the rest of the values

People continue to try fancy techniques here *wavelets, graphical models, entropy models,…*

Optimization Roadmap

Build Query Plans



Analyze Plans

1. For SFW, Joins queries
   a. Brute-force? Sort? Hash? Count?
   b. Pre-build an index? B+ tree, Hash?

2. What statistics can I keep to optimize?
   a. E.g. Selectivity of columns, values

Cost in I/O, resources?
To query, maintain?