Lecture 14:

Scale, scale, scale

(Indexing + Sorting, Hashing, Counting)

Announcements

2 guest speakers

DJ Patil, today

Theo Vassilakis, Thursday

Project 2 feedback

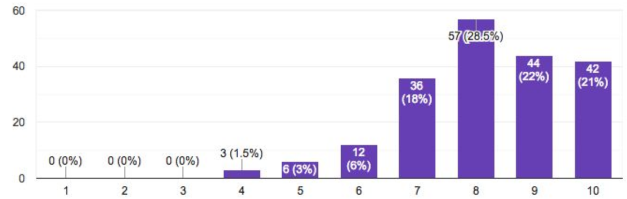
- Form: Learning experience? Repetition?
- PDFs for submission
- Motivation: Why colab? Why data visualizations? Why 1 cloud stack?

Metrics
Projects 1 and 2

(THANKS!!)

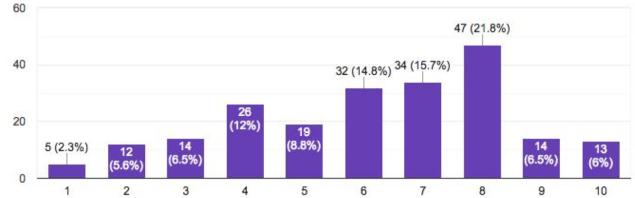
How valuable of a learning experience was this assignment?

200 responses



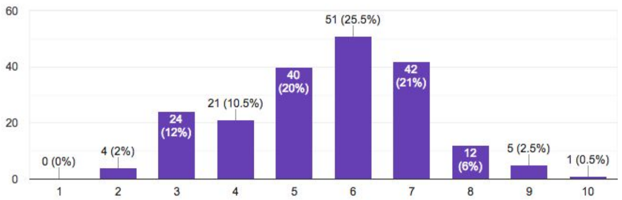
How valuable of a learning experience was this assignment?

216 responses



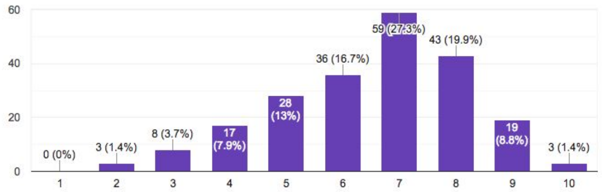
How difficult was this assignment?

200 responses



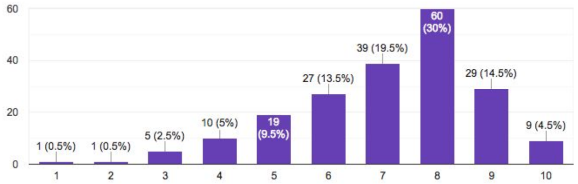
How difficult was this assignment?

216 responses



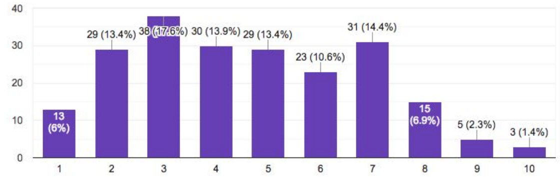
How enjoyable was the assignment?

200 responses



How enjoyable was the assignment?

216 responses



- Takeaways: P2 got too long and repetitious. PDFs from colab are a pain!
- Tweaks: No PDFs in P3. Split P2 for future.

Project

Goal #1

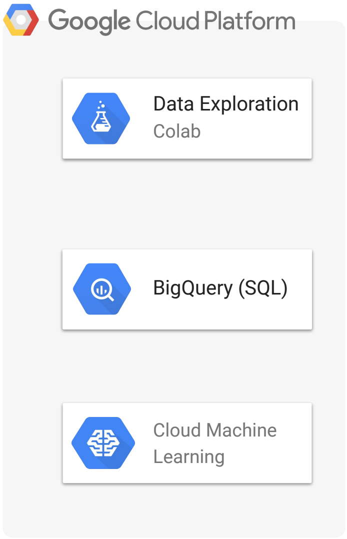
Full data cycle on cloud

(recap from Lecture 1)

1 Run queries on public datasets

2 Explore/Visualize public datasets

3 Predict using Machine Learning



Notes

- Concepts are the key. Easy to go ‘cloud hopping’
- Key goal: Do (a) deep data cycle, (b) for 10s-100s of GBs of real data

Goal #2

Sharing data cycle from 1 to teams of 10s to 1000s

(90s -)

Publish Dashboards of summary data

(2010-)

Share 'read' access to summary story with analysis+visualizations

(+ modify versions)

jupyter

Create, Share notebooks

Live code, equations, visualizations

(2015 -)

R/W Collaborate with data teams

colab

Collaborative

Co-Edit in real-time (like Goog Docs)

(based on jupyter)

Summary

When you finish P3 . . .

- Full Query-Visualize-Learn data cycle on cloud stack
  - Experience working on 10s-100s of GBs of data on machine cluster
  - Gaining an ability to grok multiple real datasets (ncaa, github, wbank, …)
  - Working with popular SQL/Visualization/Learning libraries
  - Gaining intuition of scale, performance, pain and costs in running 100s of queries

- Collaborate in small teams for r/w access to data analysis (e.g., colab)

- Hop-Skip-Jump to any cloud stack or application based on same concepts

⇒ Strong suggestion

Do a great job on P3. Link off your CV/resume.

This week

Scale, Scale, Scale

How to read/write indices?

Sorting, Counting, Hashing

(for RAM, Disk, Clusters)

B+ Tree Index Search [recap]

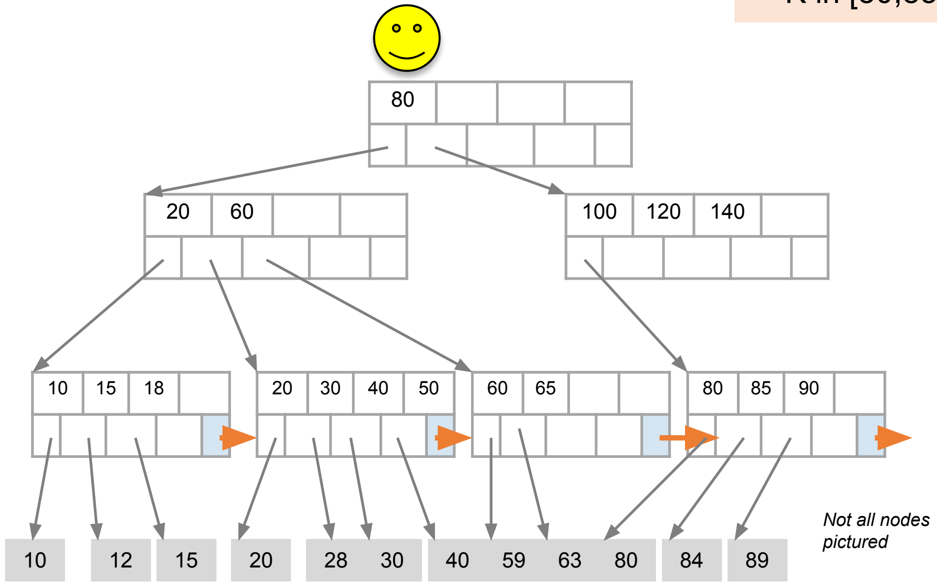
K in [30,85]?

30 < 80

30 in [20,60)

30 in [30,40)

To the data!

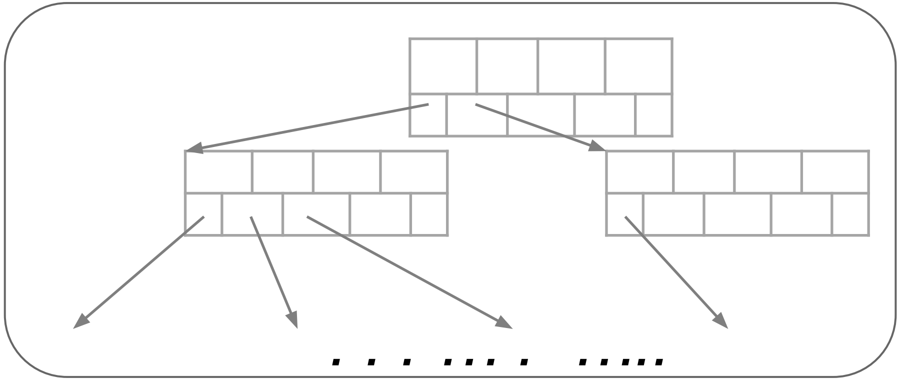


Simple Cost Model for Search

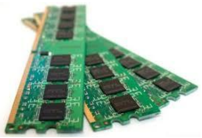
- Let:
  - ***f*** = fanout, which is **in [d+1, 2d+1]** ***(we'll assume it's constant for our cost model…)***
  - ***N*** = the total number of *pages* we need to index
  - ***F*** = fill-factor (usually ~= 2/3)
- Our B+ Tree needs to have room to index ***N / F*** pages!
  - We have the fill factor in order to leave some open slots for faster insertions
- What height (*h*) does our B+ Tree need to be?
  - h=1 → Just the root node- room to index f pages
  - h=2 → f leaf nodes- room to index f^2 pages
  - h=3 → f^2 leaf nodes- room to index f^3 pages
  - …
  - h → f^{h-1} leaf nodes- room to index f^h pages!

→ We need a B+ Tree of height h = $\left\lceil \log_f \frac{N}{F} \right\rceil$!

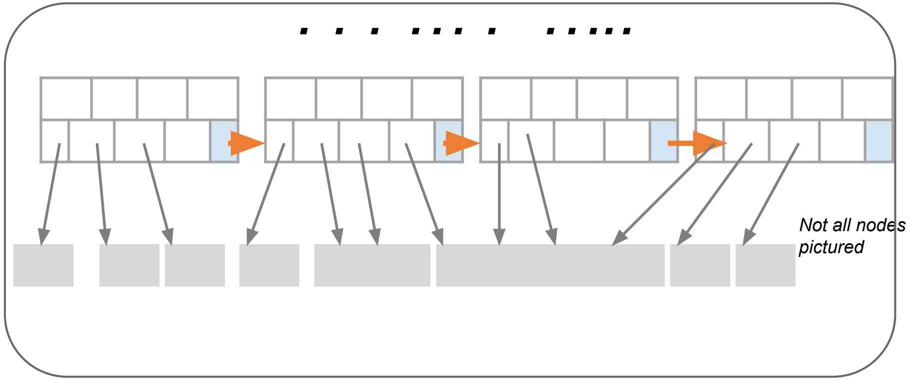
Search cost of B+ Tree (on RAM + Disk)



Read 1st levels
Into RAM buffer



Rest of index on disk



$1 + f + f^2 + f^3 + \ldots <= B$

Keep 1st L_B levels in RAM of size B

Algorithm: B+ Search
- Read 1 page per level
- Pages in RAM are free
- Read 1 page for record

IO Cost: $\left\lceil \log_f \frac{N}{F} \right\rceil - L_B + 1$

where $B \geq \sum_{l=0}^{L_B - 1} f^l$

Simple Cost Model for Search

- To do range search, we just follow the horizontal pointers

- The IO cost is that of loading additional leaf nodes we need to access + the IO cost of loading each ***page*** of the results- we phrase this as “Cost(OUT)”

IO Cost: $\left\lceil \log_f \frac{N}{F} \right\rceil - L_B + Cost(OUT)$

where $B \geq \sum_{l=0}^{L_B - 1} f^l$

Fast Insertions & Self-Balancing

- We won’t go into specifics of B+ Tree insertion algorithm, but has several attractive qualities:
  - **~ Same cost as exact search**
  - ***Self-balancing:*** B+ Tree remains **balanced** (with respect to height) even after insert

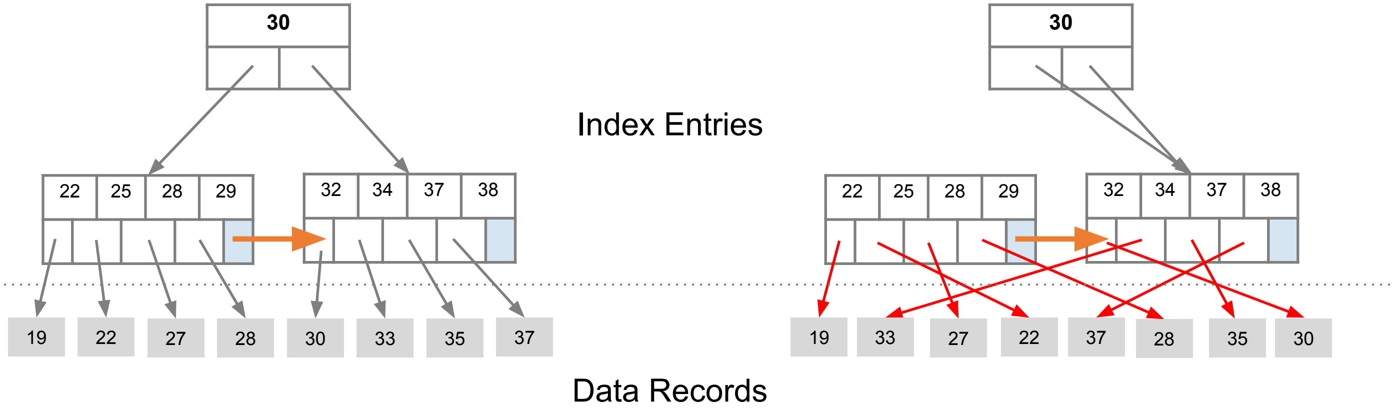
B+ Trees also (relatively) fast for single insertions!
However, bottleneck if many insertions (if fill-factor slack is used up…)

Clustered Indexes

An index is ***clustered*** if the underlying data is ordered in the same way as the index's data entries.

Clustered vs. Unclustered Index

E.g. Person(name, age, SSN)



Clustered

(e.g., Sorted by SSN, Index on SSN)

Unclustered

(e.g. index for age)

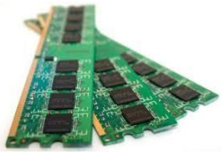
Clustered vs. Unclustered Index

- Recall that for a disk with block access, **sequential IO is much faster than random IO**
- For exact search, no difference between clustered / unclustered
- For range search over R values: difference between **1 random IO + R sequential IO**, and **R random IO**:
  - A random IO costs ~ 10ms (sequential much much faster)
  - For R = 100,000 records- **difference between ~10ms and ~17min!**

Big Scaling (with Indexes)

Roadmap

Primary data structures/algorithms

	Hashing	Sorting	Counting
	HashTables ($hash_i(x)$)	BucketSort, QuickSort MergeSort	HashTable + Counter ($hash_i$(key) --> <count>)
		MergeSortedFiles SortFiles	?????
		MergeSortedFiles SortFiles	

Recall: Hashing

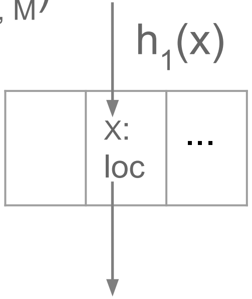
- **Magic of hashing**:
  - A hash function h_B maps into [0,B-1]
  - And maps nearly uniformly

- A hash **collision** is when x != y but $h_B(x) = h_B(y)$
  - Note however that it will **never** occur that x = y but $h_B(x) \neq h_B(y)$

- E.g., hash on any attribute A

Hashing for scale

- $h_i(x)$ % one of below
  - B = memory buf (h_B)
  - M = machines, shards (h_M)
  - <B, M> = memory buf, machines ($h_{B,\ M}$)



Algorithm: Where is record ‘x’ for read/write

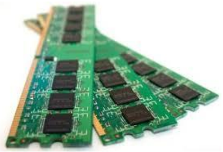
if Buf[h_i(x)] matches ‘x’ // check if really x?
Use record

- Multiple hash functions (uncorrelated to spread data)
  - $h_i(x)$, $h_{i+1}(x)$, $h_{i+2}(x)$, $h_{i+3}(x)$, . . .

Big Scaling (with Indexes)

Roadmap

Primary data structures/algorithms

	Hashing	Sorting	Counting
	HashTables ($hash_i(x)$)	BucketSort, QuickSort MergeSort	HashTable + Counter ($hash_i(key)$ --> <count>)
	Hashes for disk location ($hash_i(x)$)	MergeSortedFiles SortFiles	?????
	Hashes for machines, shards ($hash_i(x)$)	MergeSortedFiles SortFiles	

Summary

Scale with Sorting, Hashing, Counting

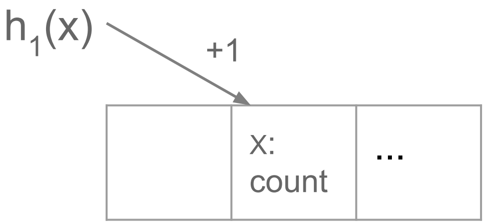
Sorting
- Indices over sorted files (B+ trees) --
  - lookups/ranges
  - Index pages with big fanout

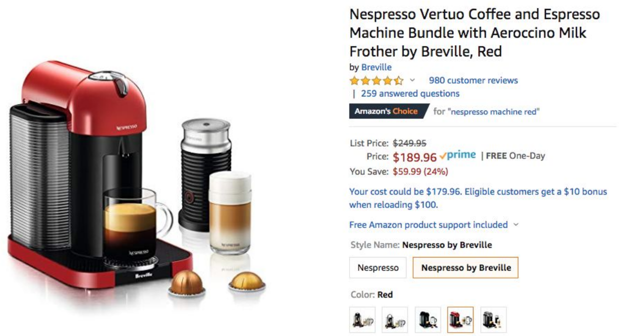
Hashing
- Hash Indices over data records

Clustered vs non-clustered indices

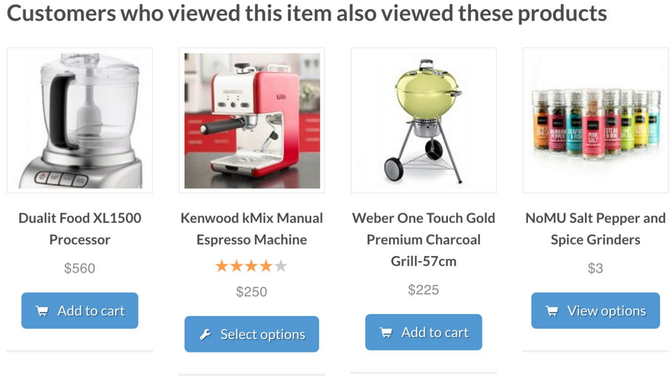
Counting?

Counting product views for billion products





Counting popular product-pairs



Problem: Number of counters = O(B*B)
(We'll discuss next class)

Hints

- most counts will be sparse
- What solves 'all' known CS problems?