Elastic Search – full text search solution
document database

Pragy

Full Text Search

Amazon Reviews – find all reviews about "product quality."
ambiguous

SQL

user-reviews

id	user-id	review-content (index)	attachment-url	rating-of-5	timest
		string			

really durable s/c
would buy again

product quality is
filmsy
⋮

Task:
find all reviews containing words "product" and "quality"
intersection

```
select * from user-reviews
where review-content like '% product %'
  and/OR? review-content like '% quality %'
limit 10
offset 100;
```

N = # rows
time = O(N)
↓
irrespective whether index is present or not

like 'product %' — can use index (prefix)

Sorted list of string

efficient

like '% product' — index(col)

index(reverse(col))

like '% product %' — neither a prefix nor a suffix query

Product was nice ①

Poor quality ②

awesome product ②

I didn't like product as it sucked ④

awesome product

I didn't like product as it sucked

Poor quality

Product was nice

index(col)

'Product %' prefix ①

'% product %' → ① ③ ④

I didn't like product as it sucked

Product was nice

awesome product

Poor quality

index(reverse(col))

'% product'

③

Full text Search

↳ Search at any location.

Document – Mongodb / couchbase / cockroach db

Key Value – Redis / Dynamodb / memcached

Wide col – Cassandra / Scylladb / Hbase

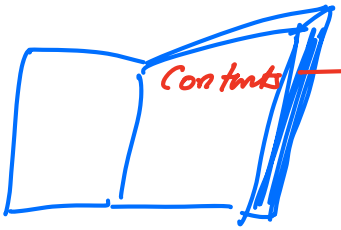
Graph – Neo4j

Vector dbs

blob – S3 / GitLFS / HDFS

solution – think of a data structure not a database

Inverse Index



index : given a page no what content does it have

Page → topic
in out

DSA

word → page no
in out

find all pages containing both 'product' & 'quality'

full-text

words	Page nos
data :	3, 5, 7, 17, 53, ...
binary search :	23, 24, 57, ...
product :	5, 11, 27, 28, 29, 33, ..
quality :	6, 11, 25, 26, 27, 53 ..

dictionary

① What db would be used to store this kind of inverse index?

inverse / inverted / reverse

Key Value
↳ list of page
↳ word

✗ incorrect
why?

Building the Inverted Index

NLP : Natural language processing.

~8% of all English words is 'the'
↳ filler word.

'to be able to understand the text'

Synonyms — multiple words mean the same thing!

lead — to lead someone
— verb
— material

Cleaning before building the inverted index

0. Tokenization
   - Split the text into 'tokens' & words
1. Stop word removal
   - remove any fillers words the an a if of and ... any bad words slave fuck ---
   - I, this, for, my, who, the, is, he, a, is, at, to, from...
2. Stemming / Lemmatization. (NLP pipeline)
   - reduce a word to its 'stem' in a dumb manner

Cards → card

```
if (words.endsWith('s')) {
    remove s from the word
}
```

-es -ing -s
-er -ion -ed -or

Cars → car ✓good
Caring → car ✗bad
lion → l ✗bad

Make a large mapping { words to 'roots'

bought → buy
husband → husband

tense / plurality / synonyms

huc abnd

Spell check.

plays → play

piano → piano

Caeio → piano

wonderful → good

fine → fine

hymns → song

husband

⋮

at the end of this code review is just a list of 'useful' words in sequence.

Build Inverted index

① 'My Kitty love this brand of cat food'

② 'My dog hate this food. It would touch it'

③ 'I love cats & dogs'

④ 'I love cats but hate dogs'

① cat, love, brand, food

② dog, hate, food, touch

③ love, cat, dog

④ love, cat, hate, dog

(1, 2)

cat → (1,1) (1,6) (3,2) (4,2)

love → (1,2) (3,1) (4,1)

brand → (1,4)

food → (1,7) (2,4)

dog → (2,1) (3,4) (5,5)

hate → (2,2) (4,4)

touch → (2,7)

How to make queries?

- imagine the user has entered a few words in the query.
- what do we want?
  1. all reviews that contain all the words in query?
  2. " " " " some " " " "
  3. all reviews ranked by 'relevance'
  4. all reviews that contain the exact sequence of the query.

Tf-idf score

$$\text{Term frequency} * \frac{1}{\text{document frequency}}$$

tf — inverse df

(1, 2)

cat → (1,1) (1,6) (3,2) (4,2)
love → (1,2) (3,1) (4,1)
brand → (1,4)
food → (1,7) (2,4)
dog → (2,1) (3,4) (5,5)
hate → (2,2) (4,4)
touch → (2,7)

query : 'love kitties'
love cat

$$\sum_{w \in Q} tf\,idf(w,d)$$

1. find all docs that match any words this query.

love → 1, 3, 4
cat → 1, 3, 4
union ⇒ 1, 3, 4

doc 1 ⇒ $tf_1(cat) * \frac{1}{df(cat)} + tf_1(love) * \frac{1}{df(love)}$

1 — $2 * 1/3 + 1 * 1/3$

doc 3 ⇒ $tf_3(cat) * \frac{1}{df(cat)} + tf_3(love) * \frac{1}{df(love)}$

0.66 — $1 * 1/3$ $\quad 1 * 1/3$

doc 4 ⇒ $tf_4(cat) * \frac{1}{df(cat)} + tf_4(love) * \frac{1}{df(love)}$

0.66 — $1 * 1/3$ $\quad 1 * 1/3$

doc 2 ⇒ 0

$tf_i(w)$ ⇒ how many time does 'w' occurs in doc i?

$df(w)$ ⇒ how many docs does this word appear in?

apple / myopia

common rare

d1 → apple

d2 → myopia

w1 → appears in literally all docs

w2 → appears in exactly 1 doc

query (w1 w2)

the more common the word is the less information it contains

less interesting it is

$tf * \log\left(\frac{1}{df}\right)$

↳ df are very large

All of the above is happening within 1 server

Apache Lucene → tf idf algorithm + inverted index Data structure } on 1 server.

Amazon has over 1 trillion reviews
& people search those reviews billions of time /day!

large data ⟶ "Sharding"

10:18 → 10:25

Single server running Apache Lucene

① Single point of failure
② it will not be able to store trillion docs
③ It will not be able to handle billion search queries/day.

Sharding → how to partition across servers
= horizontal partitioning across servers

Replication

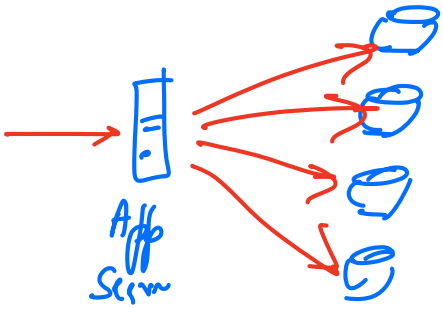
Sharding in Elastic Search

Question : Determine the sharding key

- Sharding key should be part of the query.
- data & load should be evenly distributed
- freq queries should hit 1 or atmost a few shards
  not fan-out nodes / servers
- Cardinality of key should be high
  # of distinct values
  never choose 'gender' as a sharding key

∴ Max # of servers that you can scale to ≤ Cardinality of Sharding key

Japanese hand-held fan



if you're using user-id as sharding key

2B users

⇒ 2B shards? 2B servers ✗ NO

1000 shards → each shard will store 2M users

What data are we sharding?

① the text documents — comments, reviews, posts, msgs, answers

② Inverted index (hashmap)

Query

① given doc id fetch doc

② given some text find

matching docs

③ given a word find its df

④ given a word find which docs it appears in

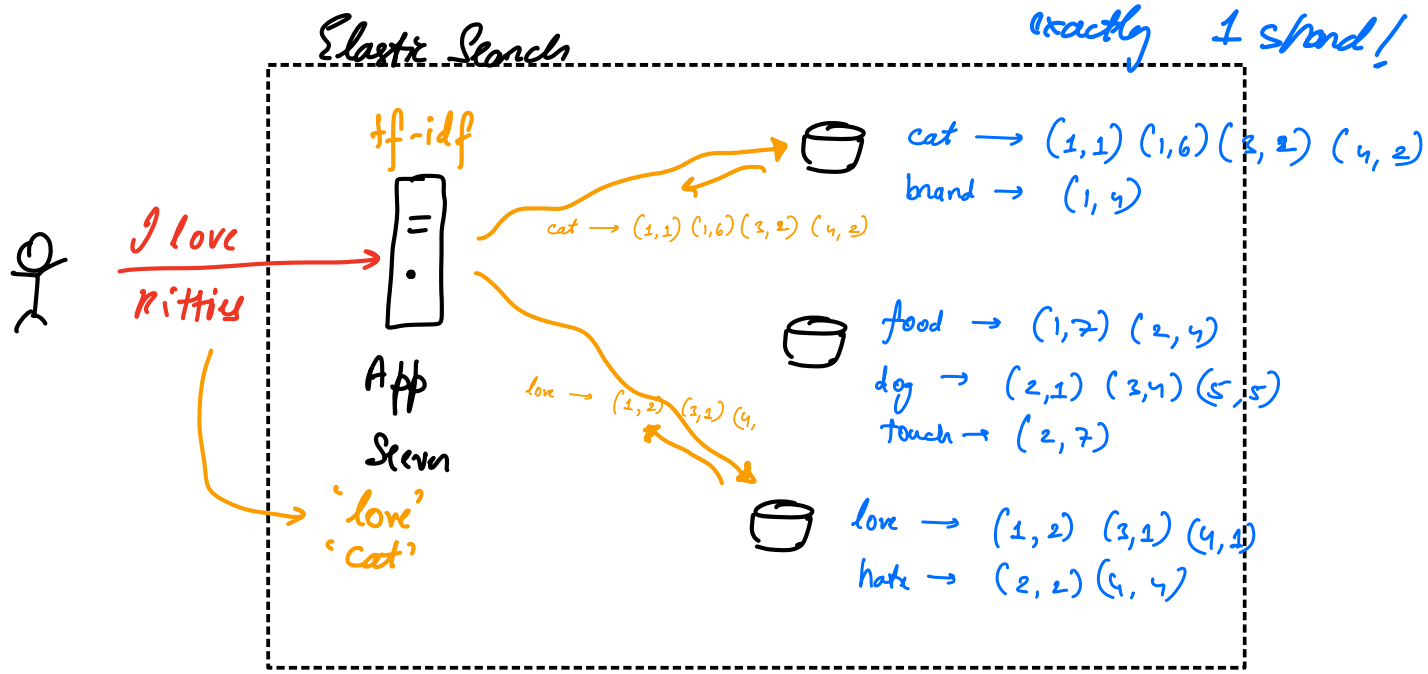
① for text docs best sharding key ⇒ doc id

② for inverted index best sharding key ⇒ ~~word (key)~~ bad!!

key - value

Attempt 1: shard the inverted index by word (key)

each word lives in exactly 1 shard!



cat → (1,1) (1,6) (3,2) (4,2)
love → (1,2) (3,1) (4,1)
brand → (1,4)
food → (1,7) (2,4)
dog → (2,1) (3,4) (5,5)
hate → (2,2) (4,4)
touch → (2,7)

① given query process it

② for each word go to shard get data for that word

③ collect all this in

app server

(3) Calculate tf idf score
(4) return the top docs

Issues

- for common words entry list is v. large

1 trillion docs
word ~ 10% of all docs
100 B docs

word → [doc id, freq], [doc id, freq] ...
8 bytes 4 bytes
100 B entries

v. v. large n/w overhead

1200 B bytes
~ 1.2 TB data

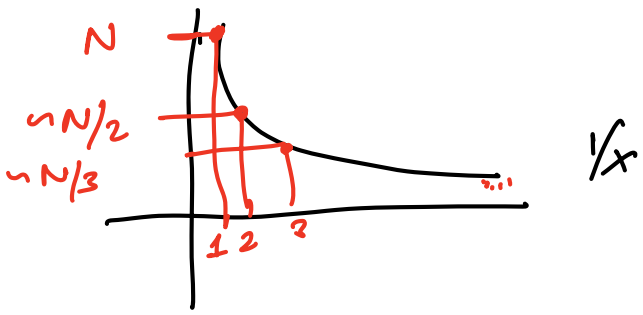
- all this massive amount of data needs to be processed in app server

Calculate tf idf
Sort by score

v. Compute intensive

- Zipf's law

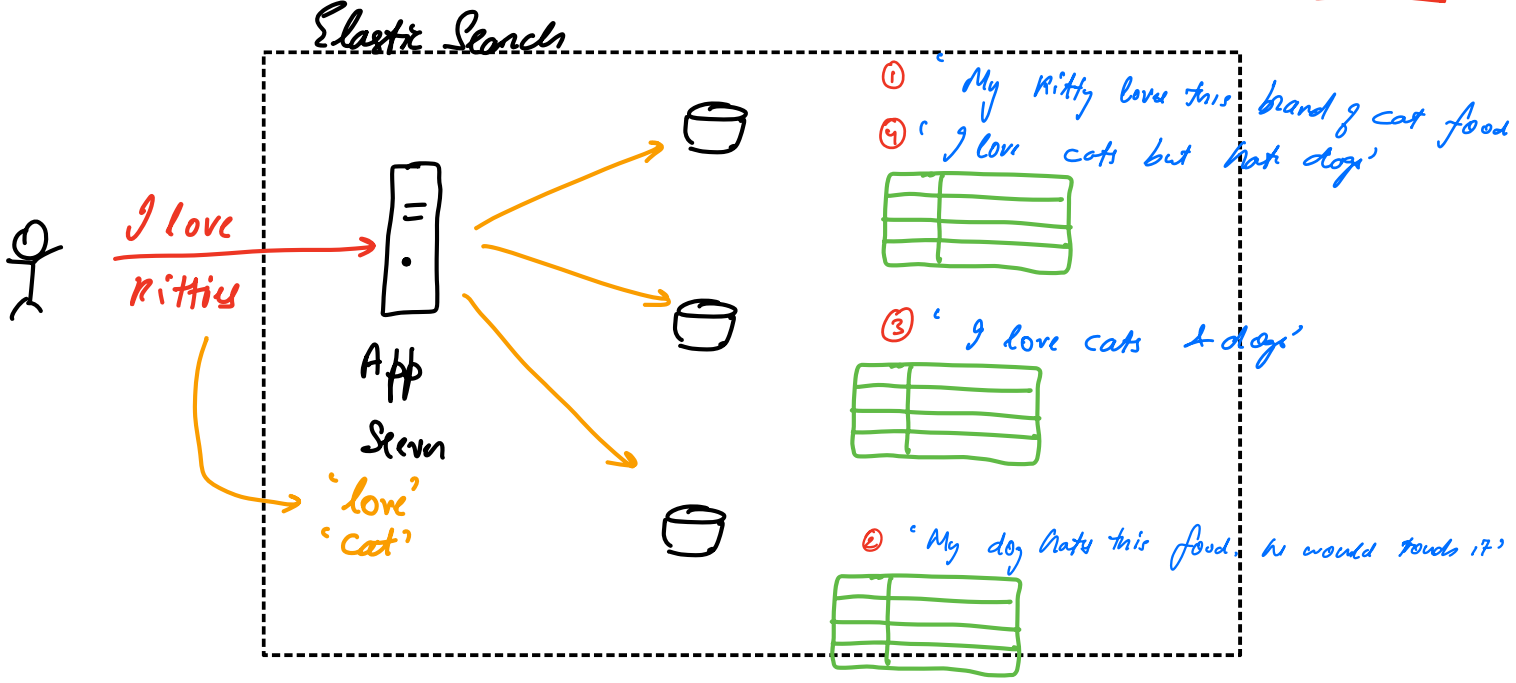
some words are very heavy
if a shard unfortunately has multiple heavy words
↓
uneven data dist.



Attempt 2 — shard by the doc id

inverted index

1 trillion docs
1000 servers
each server has ≈ 1B docs



① 'My Kitty loves this brand of cat food'

② 'My dog hates this food. It would taste it'

③ 'I love cats & dogs'

④ 'I love cats but hate dogs'

Q1 will the dict (list of words in inverted index) be different for different shards?

No! ∵ each shard has ≈ 1B docs
every word is in each shard!

Q2 Which shard contains data relevant to a query?

All of them!

for any read we should send it to all shards!??

fan-out reads?

① Why are fan out reads/writes bad?

- Q queries get converted into N · Q queries
  - ↳ # of shards hit
  - load gets multiplied
- we need to wait for all shards to respond
  - we're limited by the slowest shard
  - latencies are super high!
- what if some read/write fails?
  - fail/retry/roll back
  - latencies are v. high!
  - ∵ it is v.v. likely for at least 1 shard to be down at any given time all queries will always fail!

① when a query is forwarded to a shard what does shard return?

sharding key → doc id

top 'k' most relevant docs that match this query from this shard!

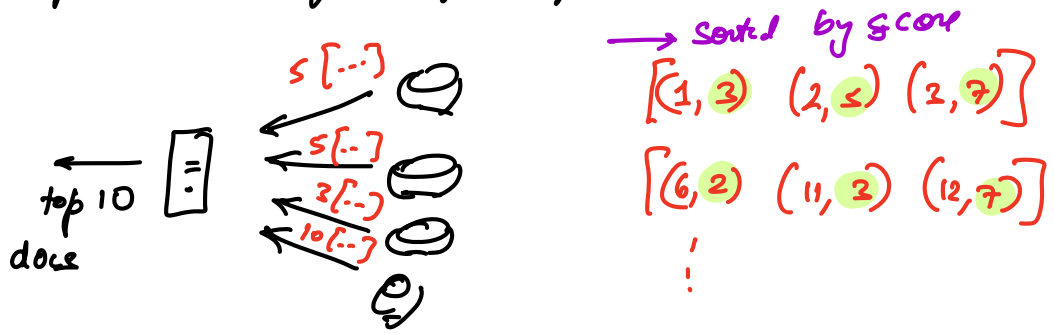
low n/w overhead

[doc id, tfidf score] ... [doc id, tfidf score]

K = 5 entries

(2) where is the major compute happening — App / db ✓ inside the shard

(3) how does app server finally respond?



fast ∵ K-way merge (min heap)

∵ tiny data (K · N entries)

↓ ↳ # of shards

# of docs returned by each shard

K = 5

N = 1000 K·N = 5000

(4) Do we necessarily need to hit every shard??

search 'diwali cracker ban'

10B tweets that 'match' this query.

user wants highly relevant results

↳ do not care about exact ranking

↳ care about a handful of relevant results

if a relevant result is Not shown

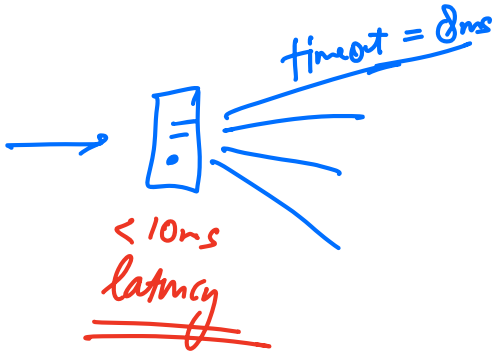
that's okay as long as whatever was shown is also super relevant

Configure for each query → how many shards to hit!!

⑤ if we send query to 20 shards
15 respond quickly
5 respond slowly

do we need to wait for slow shards? No!

we can control the latency 'exactly'!!



Sharding → docs → doc id
→ inverted index → doc id

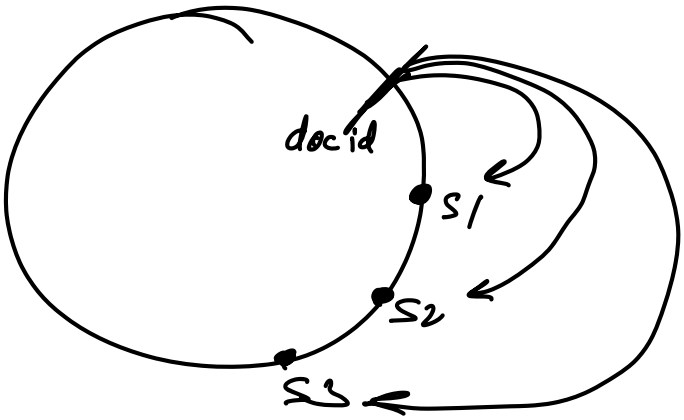
} Elastic Search is sharded by doc id

∴ Elastic Search is a doc DB!!

Replication

each doc must be in more than 1 shard

$\chi = 3$



now if we choose shards at random for query it is less likely to miss important doc

∴ each doc is in multiple shards!

Availability vs Consistency

1. Problem
2. think of what it means to be eventually consistent in this case — magic?
3. Availability / consistency

DSA

1. problem
2. brute force
2.5 observations — magic?
3. optimized

Consistency

1. No consistency → data can be lost after commit! analytics / friends view count on video
2. Eventual consistency
   - lack eventuality ↳ for some time we can have stale reads
   - ↳ eventually updated data will show up
3. Immediate
   - ↳ up to date data always

What does Eventually Consistent in context of Search?

① all relevant docs might not show up in 1 result but if we do the same search multiple times all relevant docs will eventually show up in some result.

dist system ← Amazon review / Twitter search / ... ✓ acceptable

Elastic Search

Slack search / WhatsApp search / Email search ✗ not acceptable!

↓

small data within a single server ✓

Apache Lucene

1 server with multiple read replicas

write → all replicas to ensure consistency!

100,000 users

each msg is 1 KB

5 TB you can have $\frac{5 \times 10^{12}}{10^3}$ msgs

$= 5 \times 10^9$ msgs

$\frac{5 \times 10^9}{10^5}$ msg/user $= 5 \times 10^4$

Company with 100,000+ emp where each emp has sent > 50,000 msg?

as of Feb 2024

1 server that fits on your table

Typical server

Max server that money can buy.

CPU 2 – 16

SSD 512 GB – 8 TB

RAM 1 GB – 64 GB

N/W 100 Mbps – 10 Gbps

50,000 Rs – 2 lakh Rs

CPU ≈ 400 cores

SSD ≈ 4 PB (4000 TB)

RAM ≈ 12 TB

N/W ≈ 10 Tbps

200,000 $
1.5 crore Rs

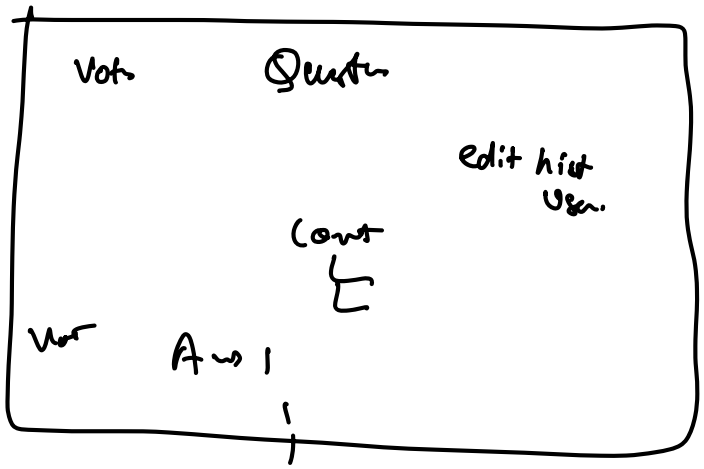
Chat GPT

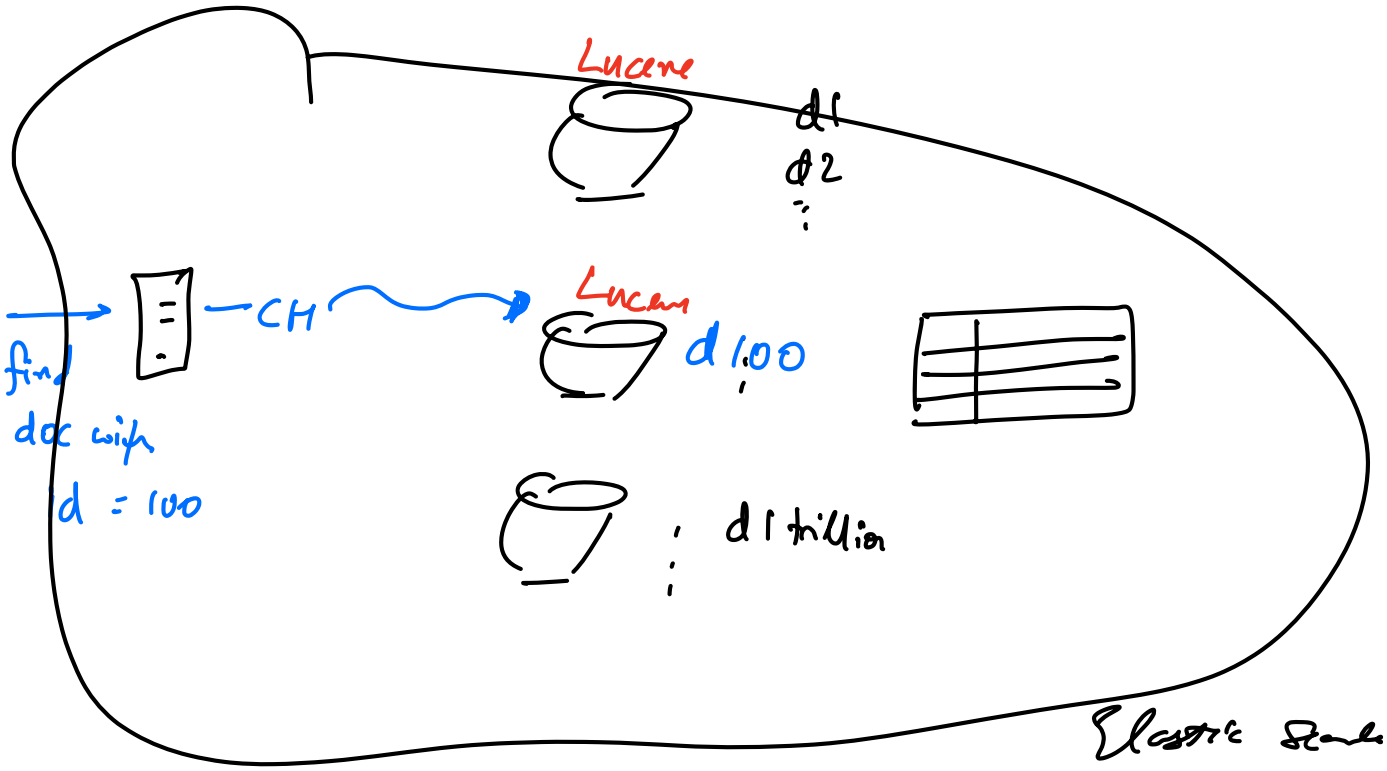
NSE/ Stock exchange server

Stackoverflow — SQL

100 TB RAM based Cache

denormalized.





Partitioning

Split the data into multiple 'parts'

Users

id	name	phone	password	email	age	gender	pic_url	nickname
1								
2								
3								
4								
⋮								
100								

① Vertical Partitioning (Normalization)

Users

id	password	email

Profiles

user-id	name	phone	age	gender	pic_url	nickname

② Horizontal Partitioning.

India

Users_India

id	name	phone	password	email	age	gender	pic_url	nickname
1								
2								
⋮								

lg(n) is small but not on disk

disk seek ≈ 10ms

Horizontal partition

USA

Users_USA

3								
100								
⋮								

When you horizontally partition data & you distribute these partitions across multiple servers

|||

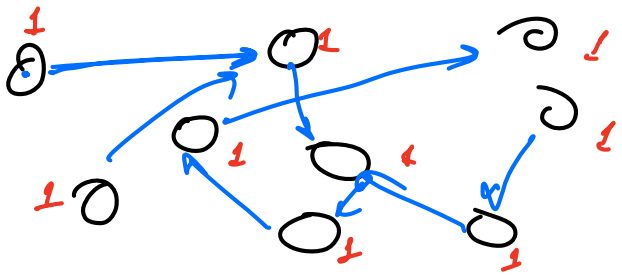
Sharding.

cleaning / inverted index / tf - idf

Information retrieval

① Adsense integration

② Pagerank algorithm



① full blown NLP pipeline (ML)

② N-grams

"This brand of cat food is awesome, my kitty absolutely loves it"

1-grams: This, brand, of, ...

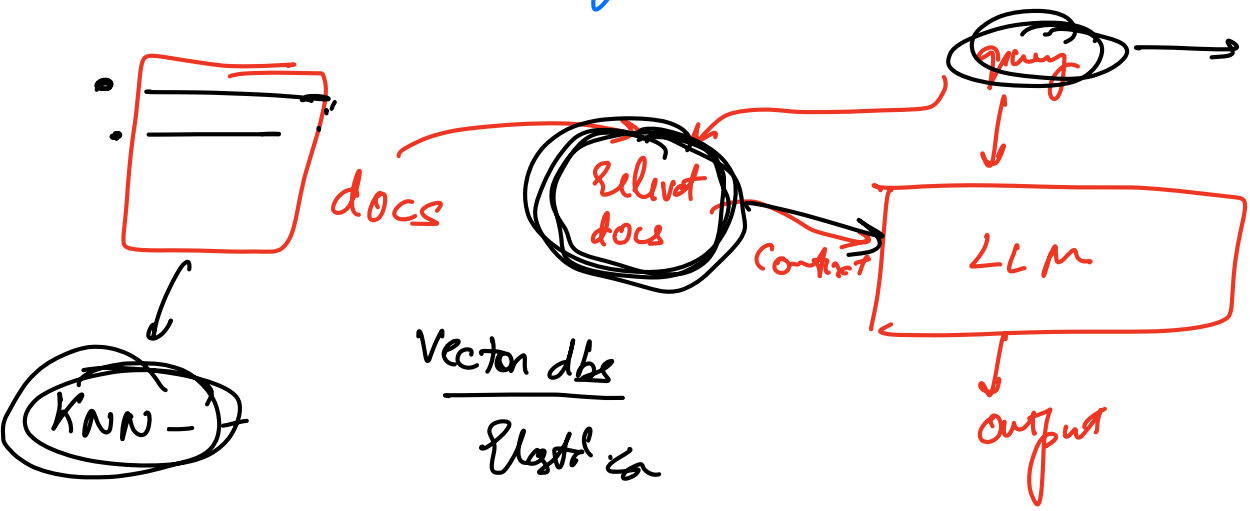
2-grams: This brand, brand of, of cat

3-grams: This brand of, brand of cat, of cat food

Retrieval Augmented Generation (RAG)

with knowledge

Gen AI



7351769221