Agenda

- Full Text Search → a bunch of docs, search query (text), find all / top documents that match the query.
- Elastic Search
- Sharding

Problem Statement

- large number of documents
- search documents (text query → search)

- Searching Logs
- Ecommerce Website
  - product title + description + attrs
  - search services
- Google Search ✗ Web Search Engine
- Social media / User created content: Tweets / quora answers / naukri.com profiles or jobs
- Scale problem descriptions
- Slack
- Google drive → search all docs
- Gmail Search

g / re / p → print
- global
- Regular Expression

Amazon Reviews → Database → SQL
→ NoSQL → Document DB

Product-reviews

id	product_id	content

add index

Select * from prod_rev
where content like query

N rows
content → M chars on avg

O(N·M) → no index

billion docs

Sorted by content
B+Tree
↳ only prefix queries

find any string containing the phrase 'value for money'

where content like '%value for money%'

index cannot be used.

find all docs that contain word [W]

trie / sorted → prefix matches

yes/no

Inverted Index → HashMap

1 → "0 g / 1 ate / 2 at / 3 this / 4 restaurant / 5 food / 6 was / 7 really / 8 nice"

2 → "0 ambience / 1 was / 2 awesome"

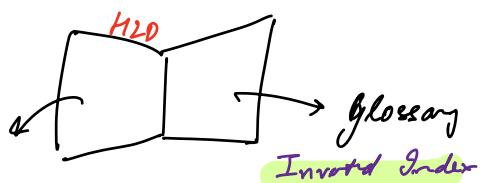
3 → "0 service / 1 sucked"

query: awesome

Key

g → [(1, 0)]
ate → [(1, 1)]
at → [(1, 2)]
this →
restaurant →
food
was → [(1, 6), (2, 1)]
really →
nice →
ambien →
awesom → all docs that contain the word awesom.
servic :
sucked
the : 18% of all English Text is the word "the"

(doc id, position)



Start

Index / Table of Contents

Chapter 1 :

Chap 2 :

load Rab : - - - - -

hashig : - - - - - -

Apache Lucene / Elastic Search / Solr / ---

① Remove 'STOP words' → useless words that appear often → eg 'the' 'a' 'an' 'of' ---

1 → " ~~a~~ / atk / ~~at~~ / ~~this~~ / restaurant / food / ~~was~~ / really / nice "

2 → " ambience / ~~was~~ / awesome "

3 → " service / sucked "

Special chars → ✗

⇓

1 → atk restaurant food really nice

2 → ambience awesom

3 → service sucked.

② Stemming / Lemmatization → takes context into account, requires ML pipeline → super slow

'develop' → [ ]
'developing' → [ ]

word → root

- e
- ing
- s
- ed
- ion
- tion

escaped → escap
escaping → escap
escape → escap

Stemming → v. fast, simple

ration → rat
Caring → Car

nation → no
↳ not a suffix

lion → l
↳ not an org

Precipitate
Pricipitation
↳ suffix

(3) Other cleaning
↳ abusive words f**k
↳ sensitive words bomb kill
↳ hate speech terrorist n**ger

Inverted Index is now built.

Search Query "The woman with the white dress"
↓ cleaning
"woman white dress"

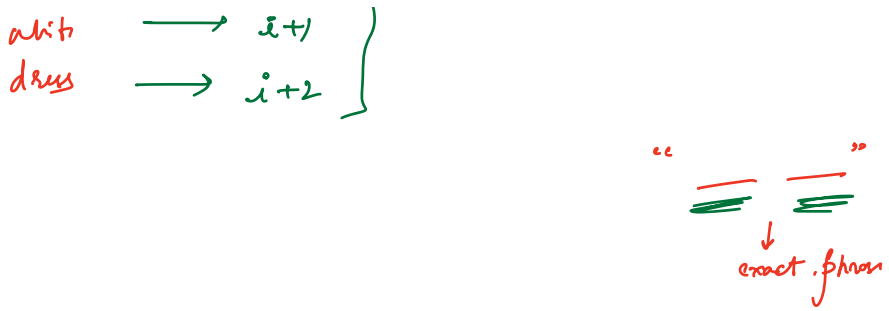
"dab"
"bcg"
"CAT"

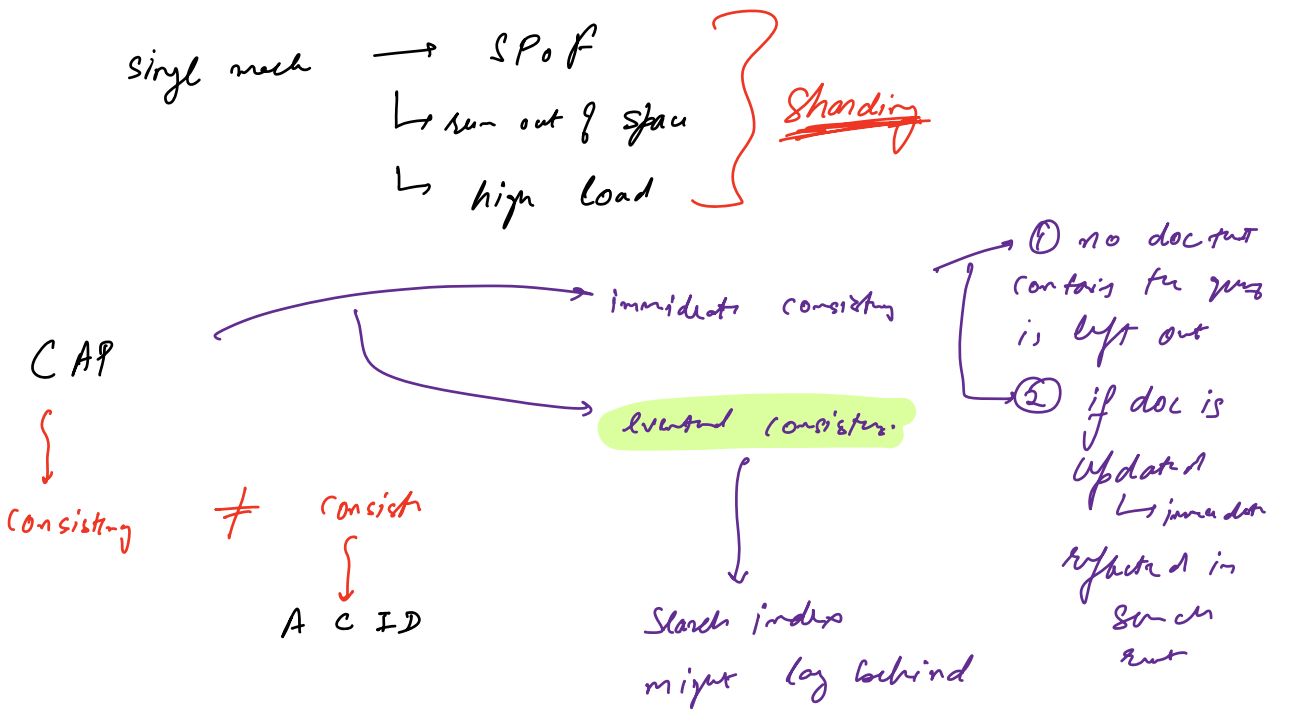
(1) all docs that contain ALL of the words
woman →
white →
dress →
} Intersection

(2) all docs that contain ANY of them
= } Union

C A T

(3) all docs with all the words in this exact order
woman → i ?


addr → i+1
dress → i+2
" "
exact phrase


single mach → SPoF
↳ run out of space
↳ high load
Sharding
CAP
Consistency
≠
consistn
A C I D
immediate consistny
eventual consistny.
① no document contains the query is left out
② if doc is Updated ↳ immediately reflected in search result
Search index might lag behind

① Shard by document-id (preferred soln) → review-id, log-id, prod-id, post-id / tweet-id

② Shard by word

insert / update / delete document → doc-id → Shard → re-index doc content.

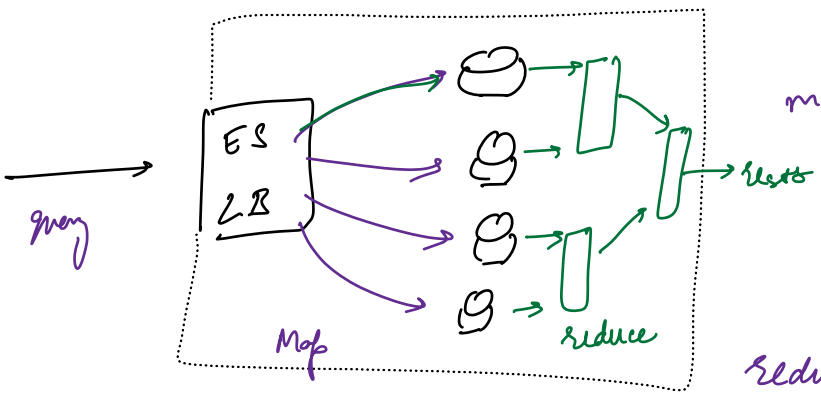
Search 'woman with dress' → go to all shards. → exact query is perform at every shard → each shard will return a list of docs that match → collect them all

Map - Reduce

items → for x in items: (Billion) → Process(x) → load all items in memory in 1 go

Lazy computation

items → map → single ——— easy to fit in memory
reduce → two item ———



map (fn, items)

↓

applying fn on each item

reduce (items, ac, fn)

reduce (nums, 0, (a,b) → a+b)

↳ reducer

reduce (strands, [], (a, b) → [... a, .. b])

concatenate / intersection

Cons of doc-id sharding

read heavy / write heavy

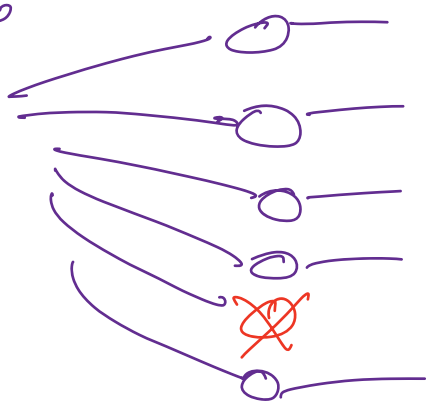
- read heavy → complex (all shards)
- write heavy → simple (1 shard)

Pros of doc-id sharding

→ latency is highly predictable. → ∵ response time doesn't depend on query

→ no need to do intersection

→ can skip unresponsive/slow shards

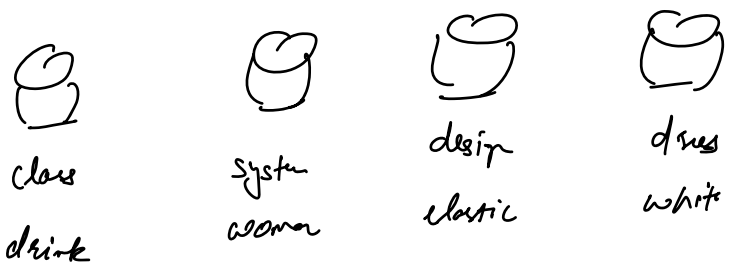


10:39 → 10:50

Open Source → Lucene / Solr / Open Search / PostgreSQL

Proprietary → Elastic Search / Algolia / Azure Search / SAP HANA - -

Word based Sharding



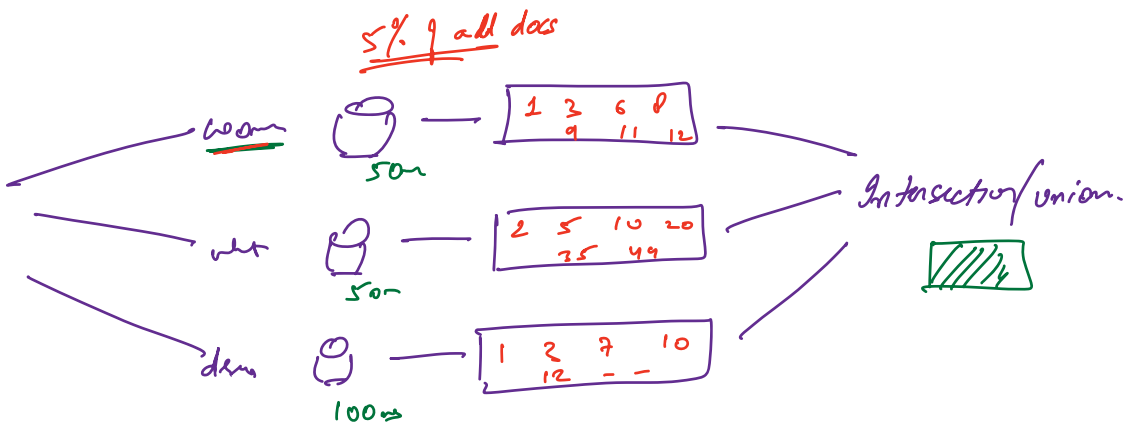
Insert / update / delete ⟶ doc = { The sun is for you and the day. }

go to all shards corresponding to the words in the doc → multiple shards

query = 'woman white dress' ⟶ go to all shards for this query.

3 words on avg.

v.v. small (< 5)

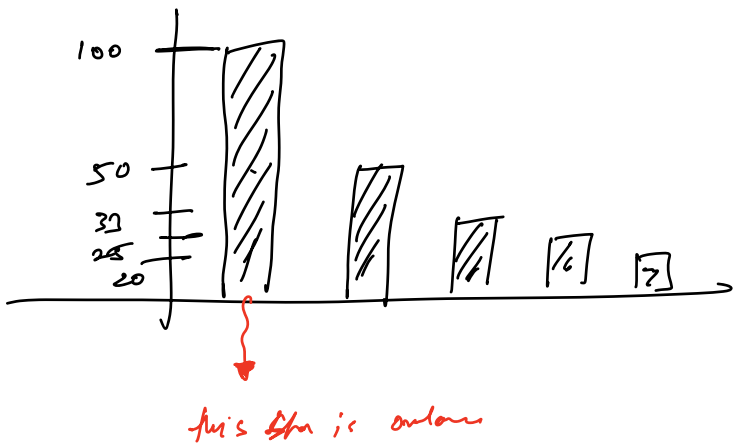


queries with simple/freq words → longer

query with more words → longer.

we need to wait for all shards (involved in query) to respond.

Zipf's law — Vsauce



ith most used word occurs with $\frac{1}{i}$ times

this shard is overloaded

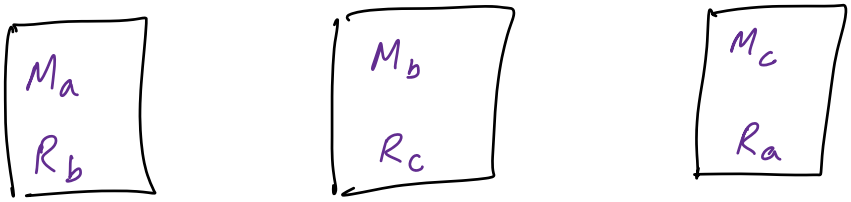
Replication

#of shards — Replication factor

N nodes → each node is a server in ES.

M shards
R replication factor. } M*R items

each server must hold M*R / N shards/replicas



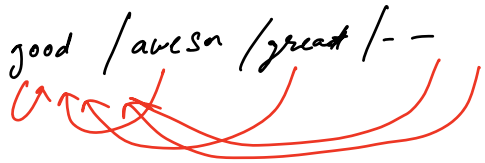
N = 3

shards = 3
D_a D_b D_c
set of docs ⇒ shard

Ensure that 1 machine only has 1 copy of any shard at most.

replication = 2

Master – Slave Replications



All docs that match a certain query

↓

Ranking

TF - IDF → Inverse document frequency

↓

Term frequency

→ 100 times

IF for quick = 3

→ d1 → quick fancy help quick fix quick fun

→ d2 → quick fox TF = 1 → 1 time

→ d3 → quick pain relief TF = 1

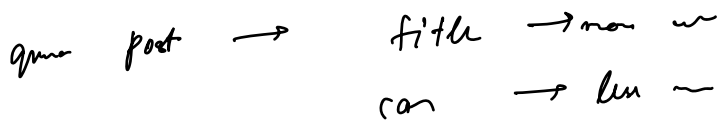
d4 → One piece

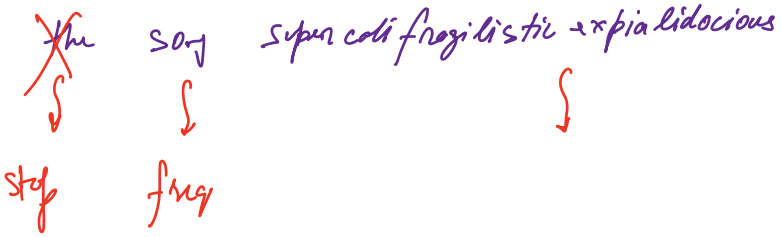
d5 → Naruto

DF (Quick) = 3 !

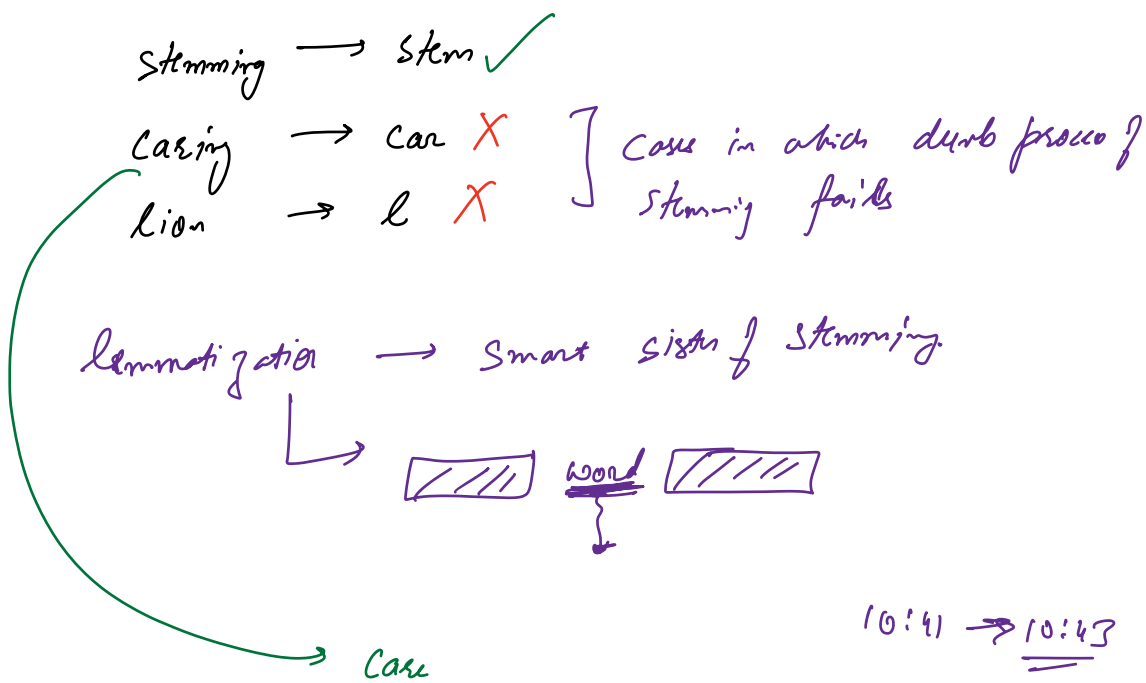
1 / DF(quick) = 1/(3+1)

3 * 1/3 1 * 1/3 1 * 1/2 log

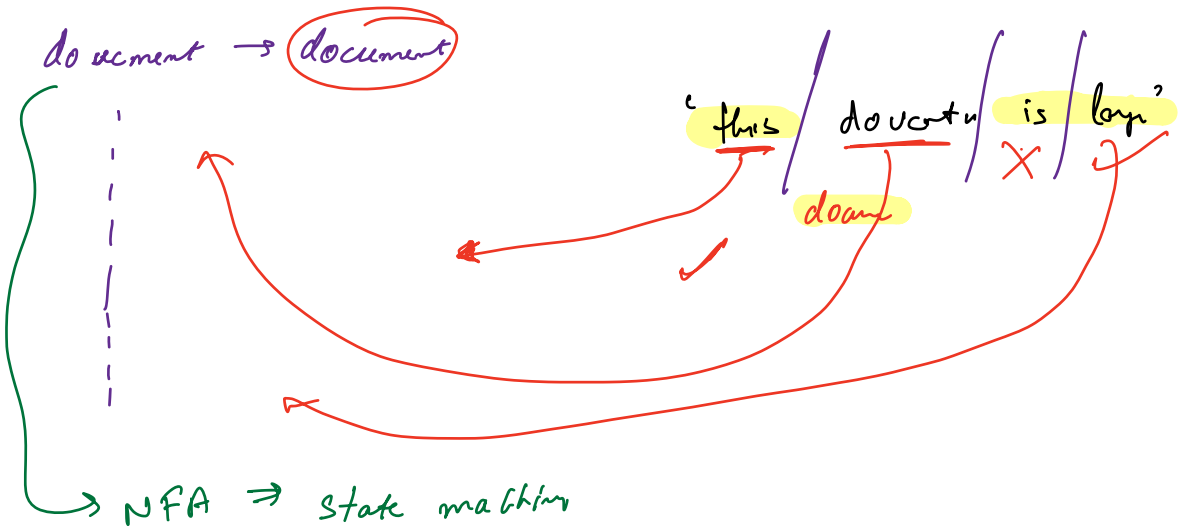


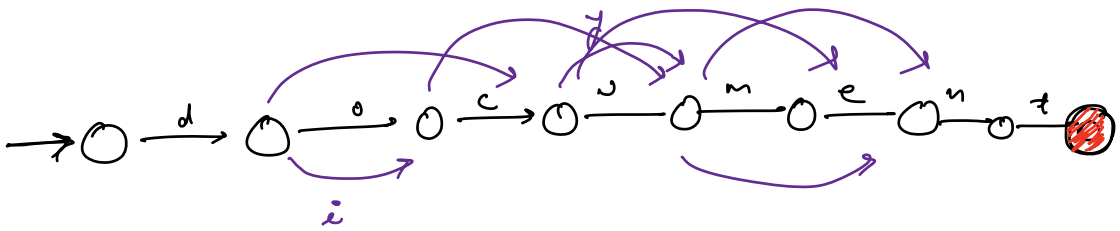


Stemming → quick & dirty & dumb process



10:41 → 10:43





Non-deterministic Finite Automata

map (fn, items) ⇒ apply this fn on each item ∈ items

reduce (items, acc, fn) (start value, reducer)

↳ (a, b) → single value

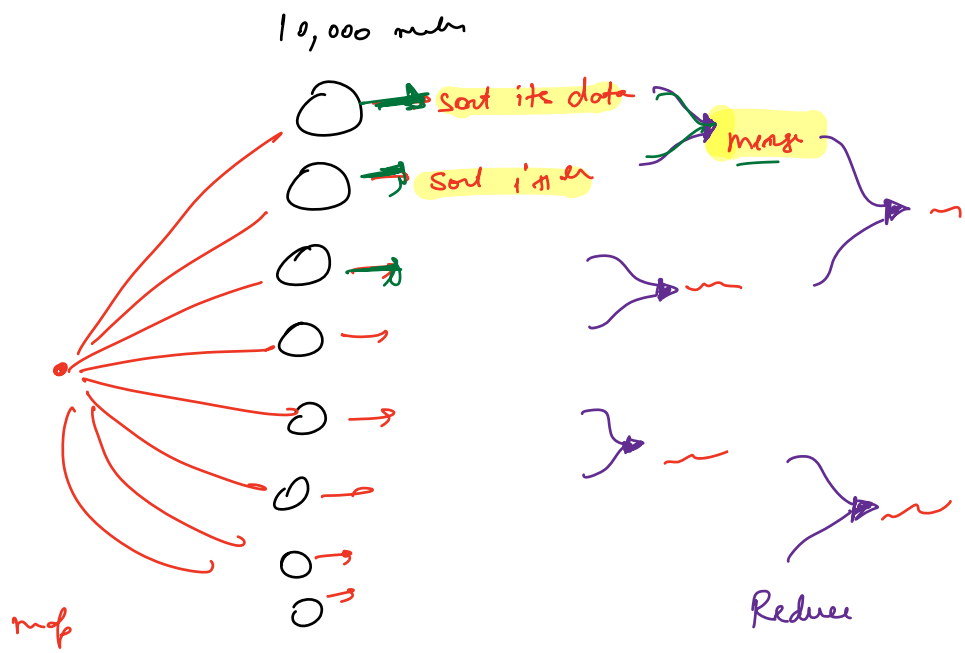
given a list of text lines → sort in lexicographic order

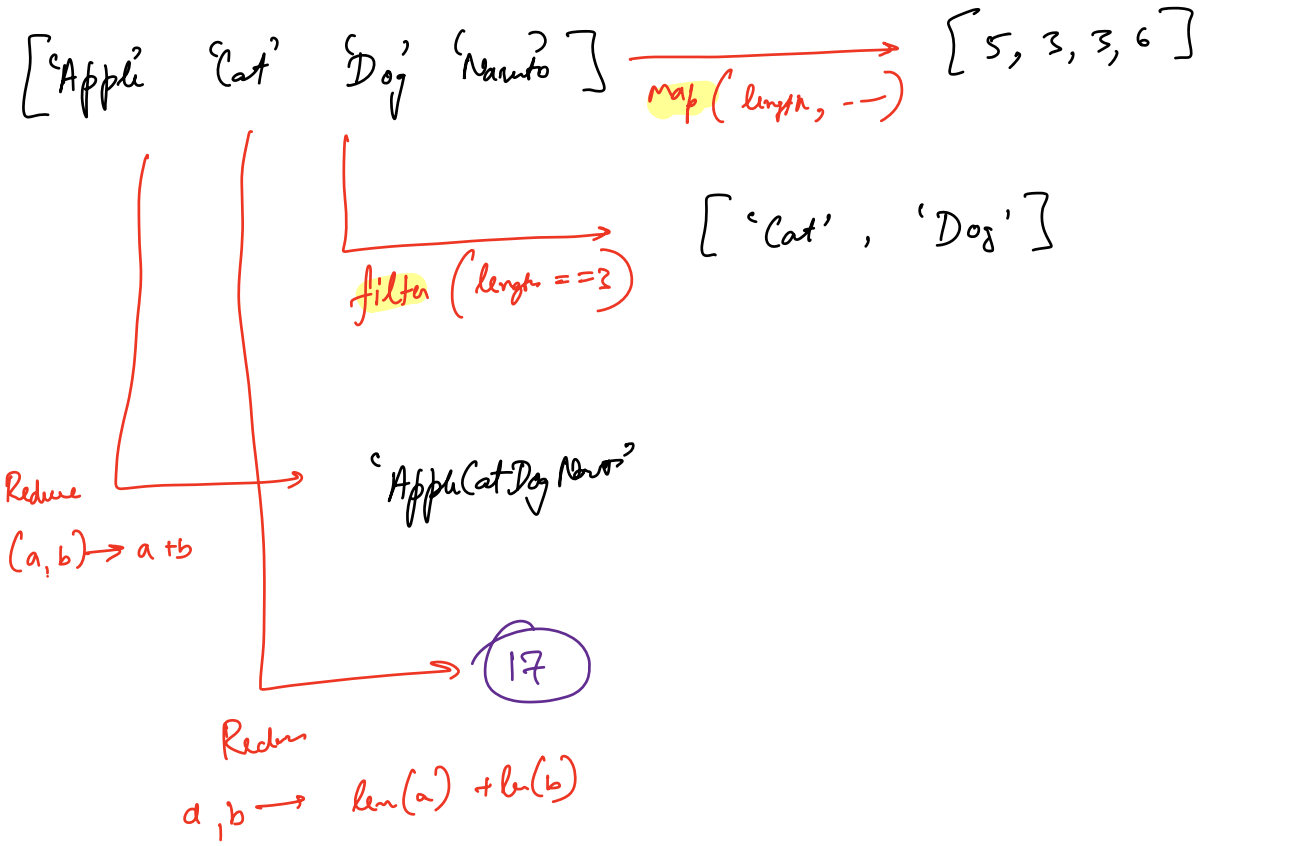
↳ 50 PB of data

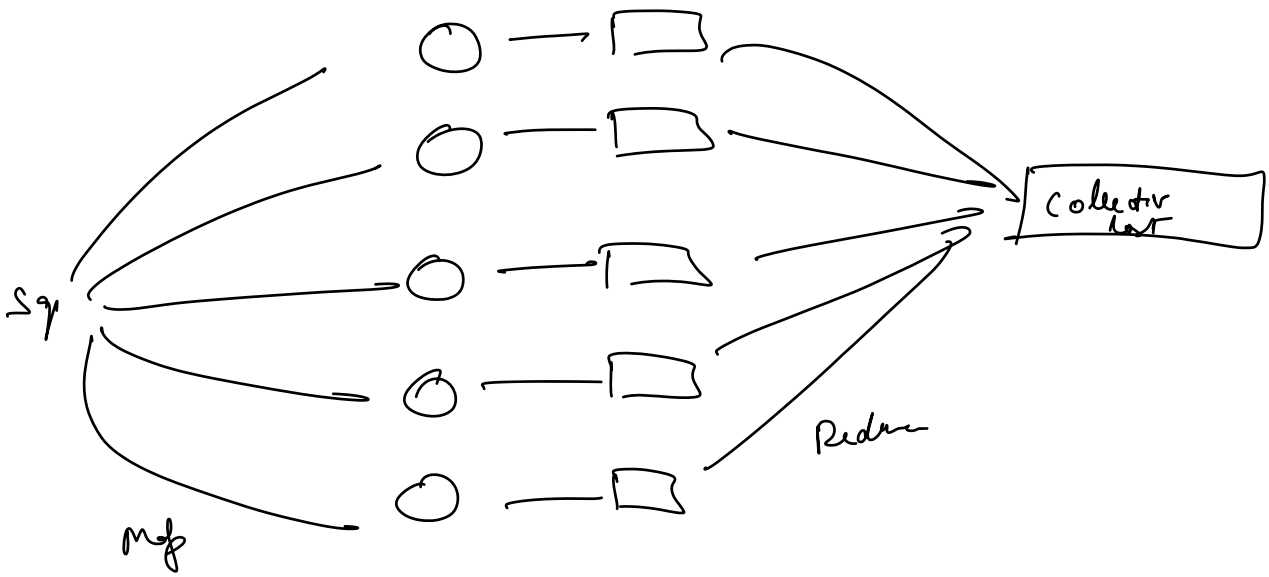
1 Byte

Kilo	1000
Mega	10^6
Giga	10^9
Tera	10^{12}
Peta	10^{15}

50,000,000 GB of data




['Apple' 'Cat' 'Dog' 'Naruto']
map (length, --)
[5, 3, 3, 6]
['Cat', 'Dog']
filter (length ==3)
Reduce
(a, b) → a + b
'AppleCatDog Naruto'
17
Reduce
a, b → len(a) + len(b)


Src
Collector
Reduce
Map

2 R+ user

user_id



d1 = 'the quick brown fox jumps over the lazy dog'

d2 = 'every shard has its own inverted index'

d3 = '

---

doc { id
field 1 : ——————
fox 2 : ——————
}

word : [(did, field_id, posn) . . . ( )]

Immidiate consistency → ES.