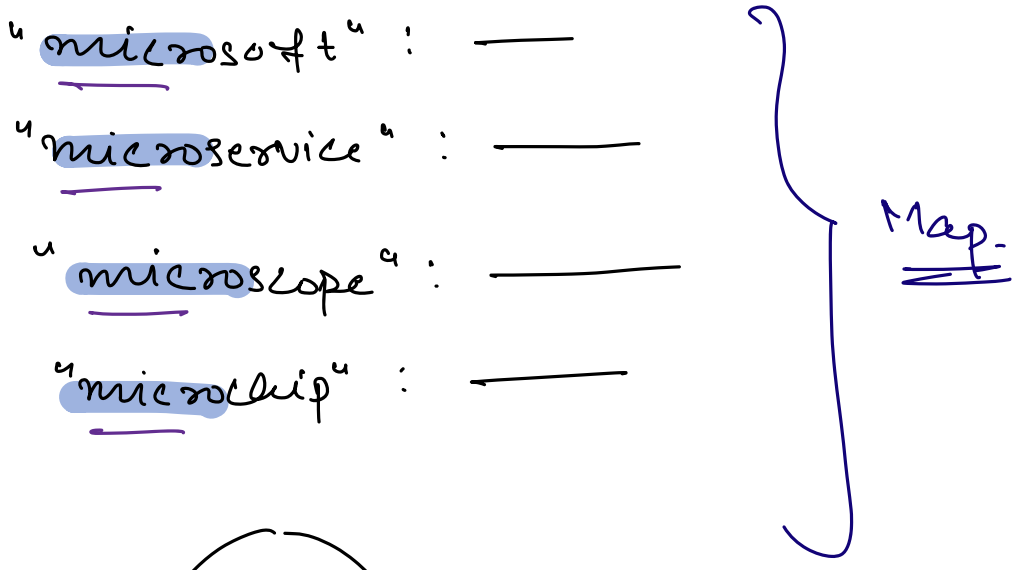
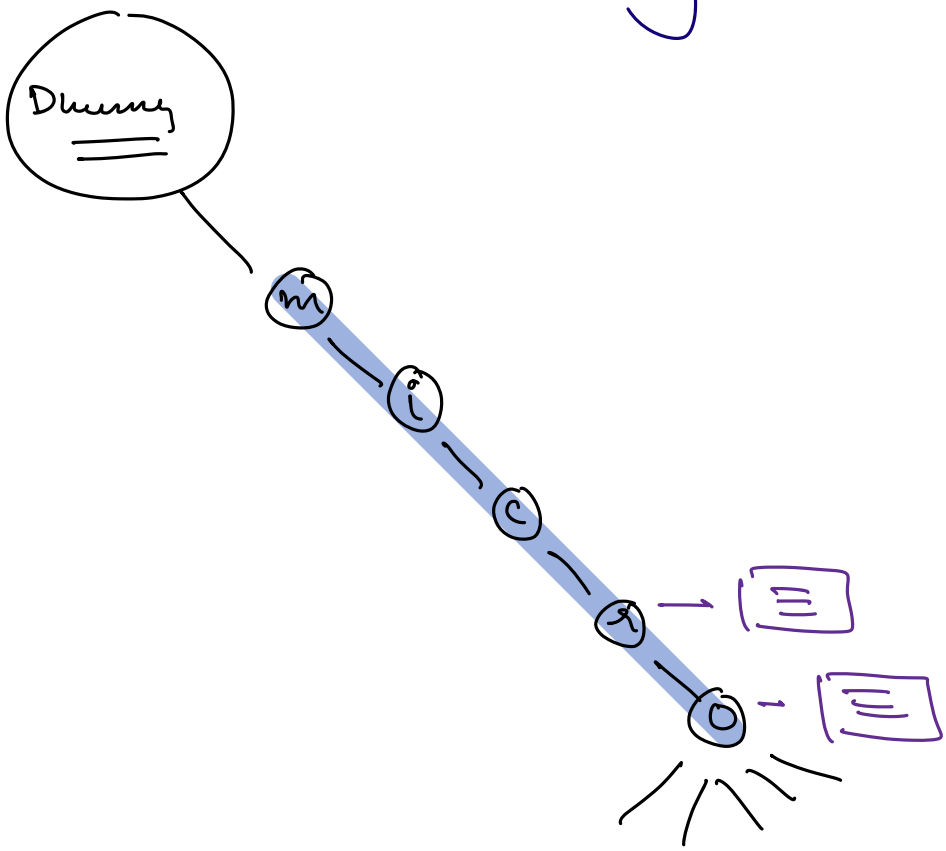
Agenda.

→ Complete Typeahead case study.

⇒



# HashMap #1

↳ K, V

- Search_query.
- frequency.

WRITE

- New Search ⇓ Insert the word in the Map with freq. 1
- Old Search ↓ ++ (increment the freq)

map.put("microsoft", 1)

↓

2 → 3 → 4 → 5

map.put("microscope", —)

getSuggestions("micr") ⇒ X

# HashMap #2

↳ prefix : Top 5 words with frequency for the given prefix.

"mic" : [ microsoft : 100K
microscope : 80K
microworld : 60K
microservice : 50K
microchip : 10K ]

microwave : 9999 → 1002 (3)

"microsoft : 100K; microscop: 80K; - - - - - -"

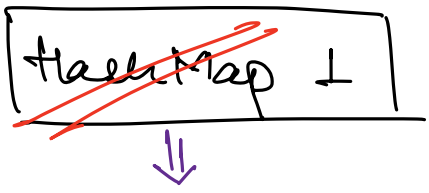
K:V DB's : Redis

String : String.

⇒ DB's like Redis supports Sharding out of Box.

# Approach.

REDIS




search_query : frequency.

REDIS.


prefix : top 5 words starting with the given prefix.

# WRITE

↳ search query.

microscope. → 80k.

# HashMap 1

1) ↳ Check if the query word is already present in the HashMap 1 or not.

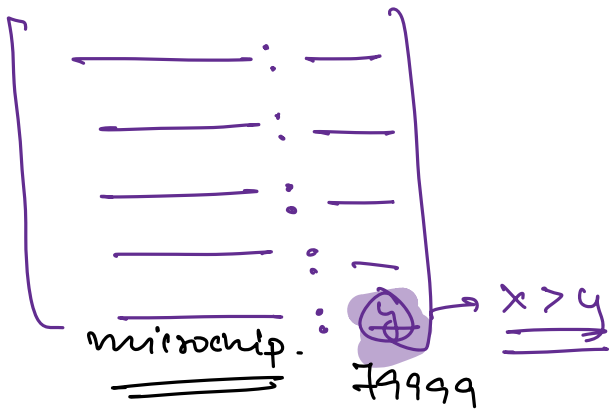
⇒ If Yes, (+1)

⇒ If No, add the entry with frequency (1) in HM1.

HM2 microscope

↳ find all the prefix strings of length >= 3.

mic
micr
micro
micros
microsc
microsco
microscop
microscope




microsoft

# READ.

⇒ getSuggestions("mic")

HM2

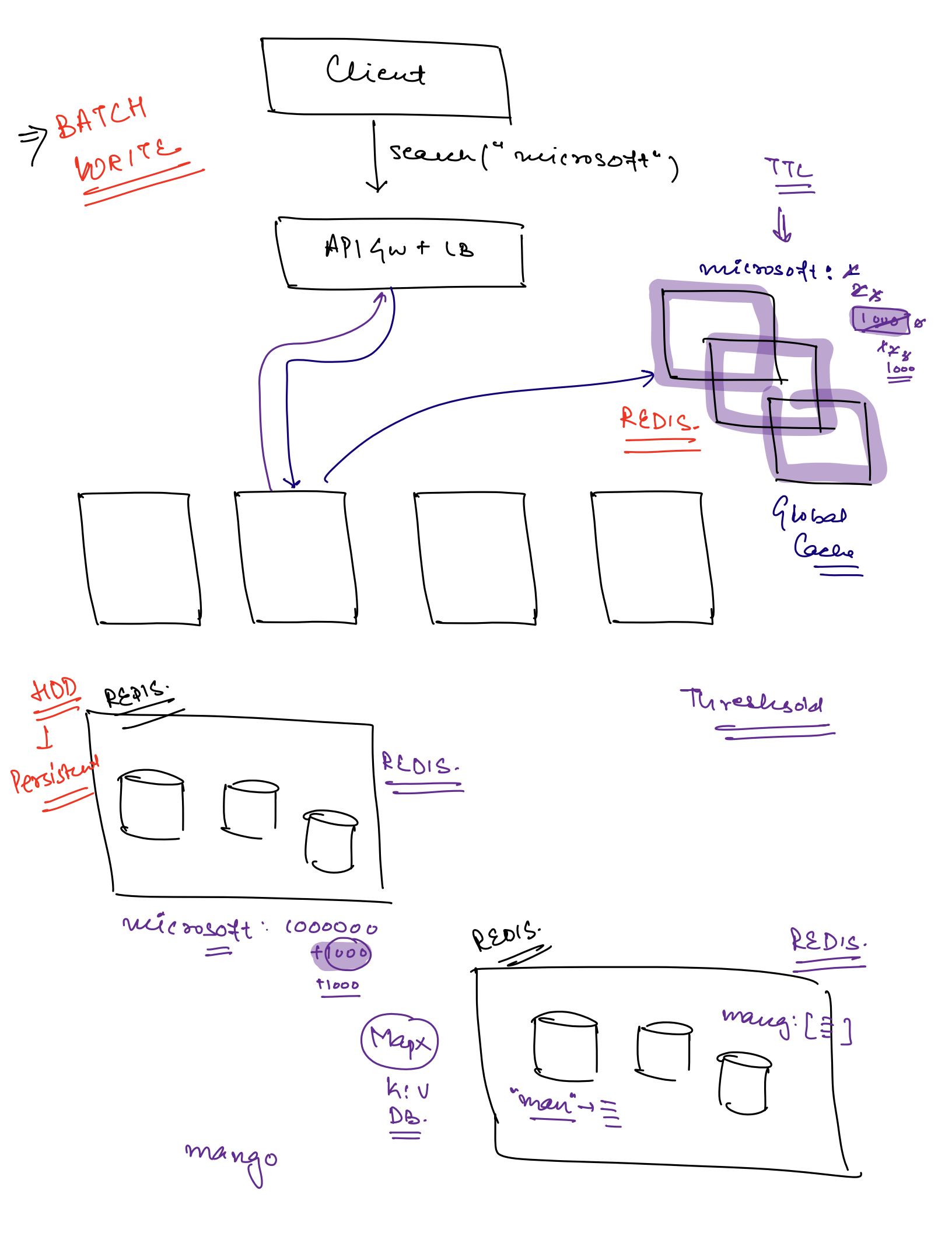
map2.get("mic")

microscope : 10000000

microservice : 10000001

⇒ low latency + Eventually consistent.

⇒ Instead of writing every single operation in DB, can we write in Batches.


Client
search("microsoft")
API GW + LB
BATCH WRITE
TTL
microsoft: x
2x
1000
1x
1000
REDIS.
Global Cache
HOD
Persistent
REDIS.
REDIS.
Threshold
microsoft: 1000000
+1000
+1000
Mapx
K: V
DB.
mango
REDIS.
REDIS.
mang: [ ]
"man" → ≡

⇒ Network B/w.

Write Qps = 200K

Read Qps = 1M

⇒ find all the prefixes of length >= 3 & update top 5 words in HM2 if required.

# of times we are updating HM1 & HM2

$$\frac{200,000}{1000} \Rightarrow 200$$

HM1, HM2 = Any K:V DB like REDIS.

⇒ DB2 | HM2 | REDIS-2

"mic": [ = ] ← microsoft.

"mis": [ = ]

⇒ READ.


# Typeahead System Design

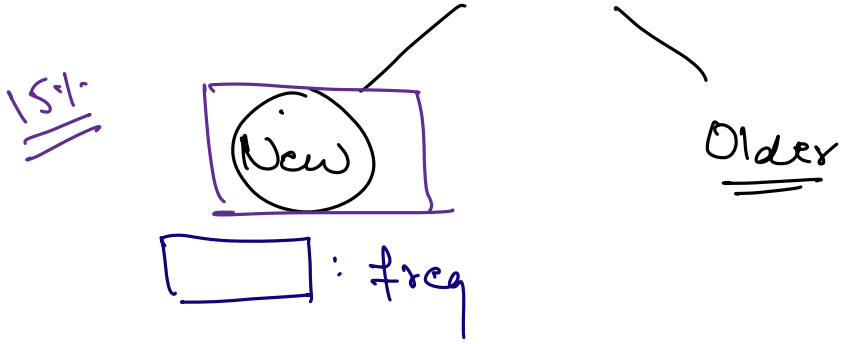
1) MVP
2) Scale Estimation ≡ Guesstimation.
3) Design Tradeoff.
4) Design Deep Dive
   - API
   - Data flow.

⇒ Write Qps = 200K.

Read Qps = 200K × 5 = 1M

⇒ Storage requirement

20B writes per day.



# Highly Available & eventually Consistent.

# Super low latency.

# getSuggestions(prefix, limit)

updateFreq(query).

# Trie

↳ Backtracking ✗

# Trie + HM at each Node to store the top 5 words.

✓ | Storage requirement ↑ | Sharding a Trie is difficult.

# K:V DB ⇒ Storing the freq of every word.

DB1 ⇒

+

DB2 ⇒ K:V DB ⇒ for every prefix, store top 5 words.

↑

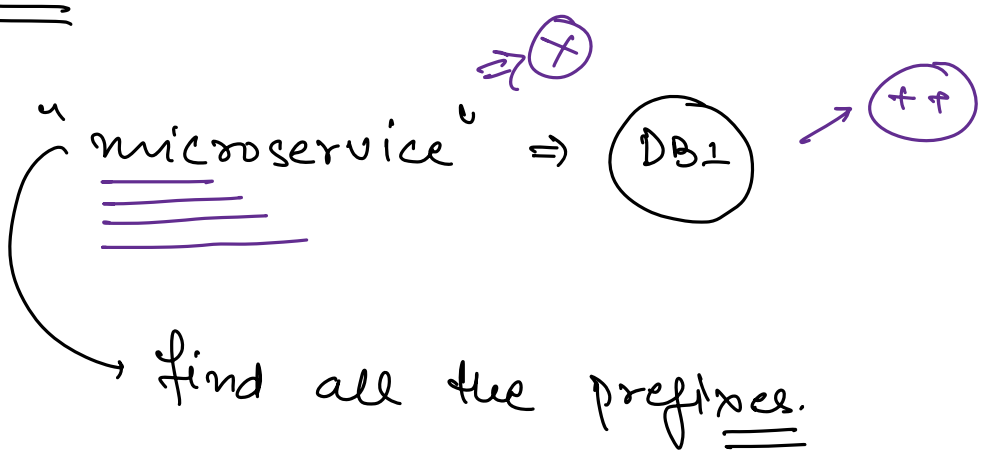
Optimise Read Queries.

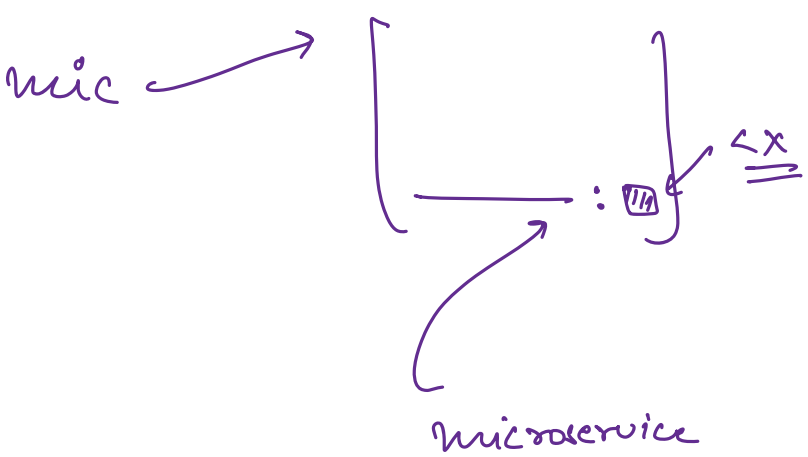
"mic" : [ microsoft : 100K ;
microscope : 80K ,
]

# READ.

getSuggestions("mic")


# Write



# Batch Writes.

# Time Decay.