Agenda

1. Invalidation & Eviction
2. Local Caching - Case study 1 → Scaler's Code Judge
3. Global Caching - Case study 2
4. Case study 3 - Facebook News Feed

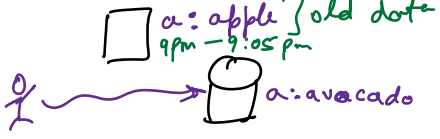
support@scaler.com

9 - 11:30 class
+10 mins doubt

Most applications are read heavy → caching.

Challenges with Caching

- Cache is not the source of truth
  - ↳ database
- Cache data can get stale



Cache Invalidation

↳ fix stale data

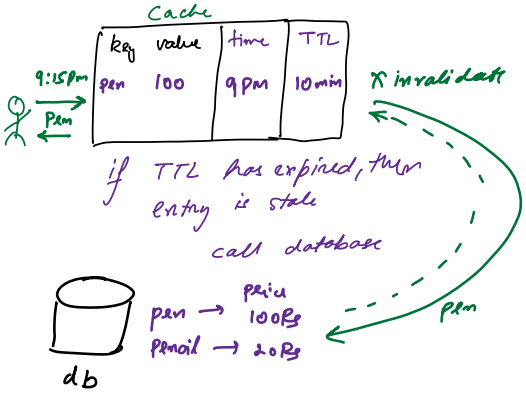
Time to Live (TTL)

Simple / Configure TTL

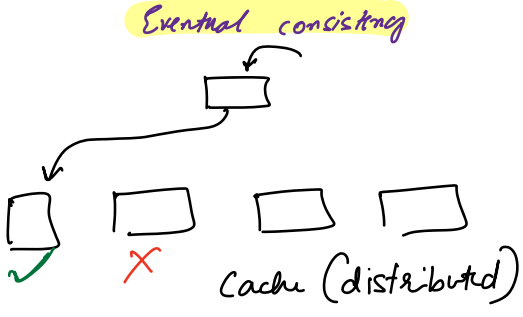
you will get slightly stale data

Write Through

- Small in size
- Cache miss → data comes in there in cache
  - ↳ lead to slowdown
- to load new data cache needs to remove existing data.
  - ↳ what data to remove?



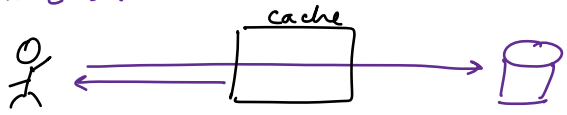
Write Back

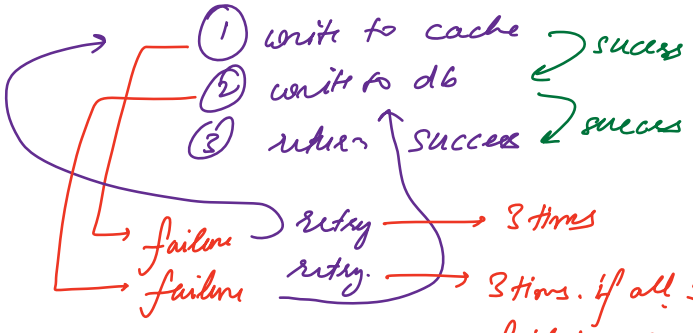


Write Around

Write Through Cache

Any updates must first be written to cache. } Synchronously (on after other) to both cache & db





? Can cache data be stale in case of Write Through Cache?

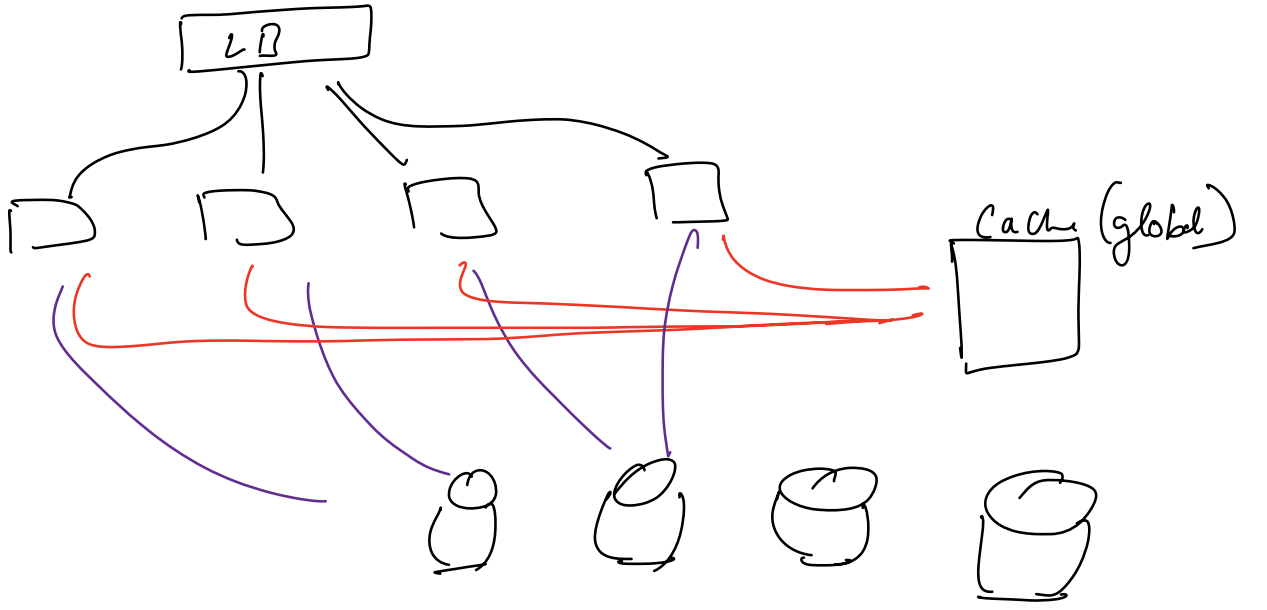
Never!!

1. Always fresh data (in cache)
2. Reads are super fast → every cache hit will return directly from cache
3. writes are much slower

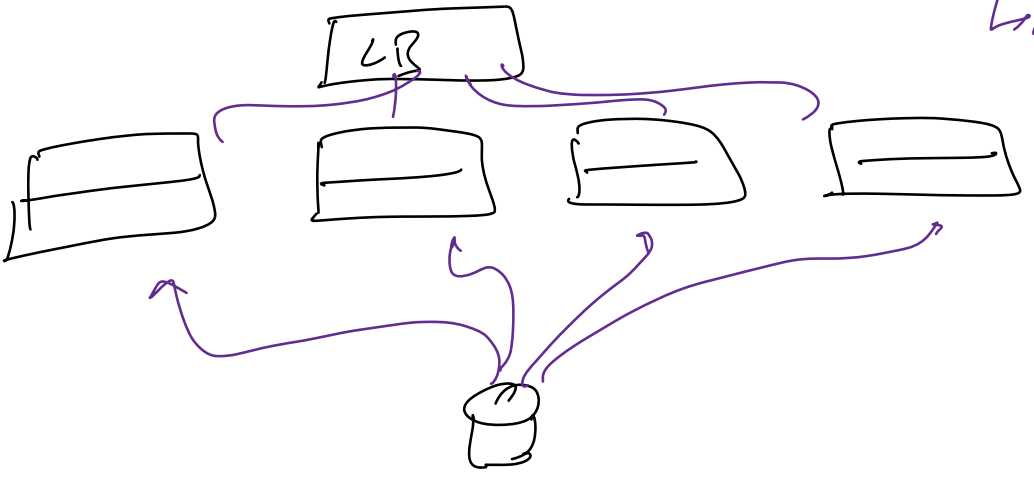
Write through cache is good for a read-heavy system.

& strong consistency is needed.

most req. are reads & not writes
90% of applications

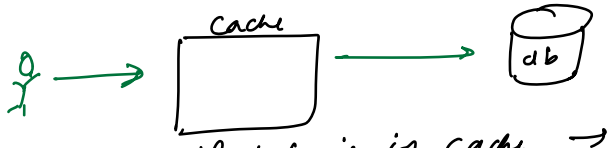


bust/invalidate
↳ destroy



Reads are fast when using a cache

↳ ∵ of how cache works.



if data is in cache → 1 ms → return from cache.

if data is not in cache → read from database ≈ 10ms

hit → 1ms → 99% cache hit

miss → 1ms + 10ms = 11ms → 1% miss

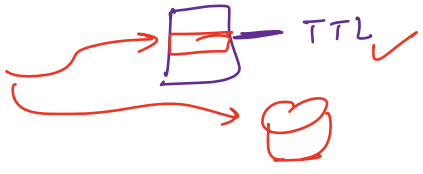
cache db

avg. time per query.

$0.99 * (1ms) + 0.01 * (11ms)$

$= 0.99ms + 0.11ms$

$= 1.1ms$



Write around cache

→ write happens directly at the database

↳ the cache might be out of sync for a short while

→ cron job (10 mins) that queries recently updated data from db & updates in cache.

```
select * from product_price
where updated_at > (now() - 10 mins)
```

→ update in cache.

eventual consistency

write-back cache (inconsistent)

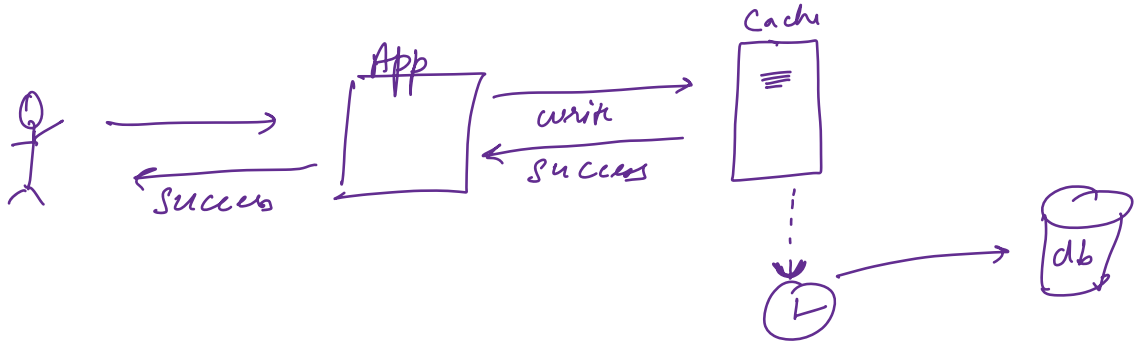
↳ first write data in cache

↳ return success

even though data is not on db!!

↳ chronical

↳ cron job that will periodically flush this data onto the database.



Cons

if cache server crashes we lose some data!!

lose

Pros

Extremely high write throughput

Use write-back cache in write-heavy system only if consistency is not important

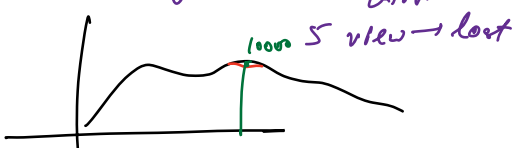
Analytics / tracking clicks on ads

↳ 10,000 views

↳ only 9995 views got written

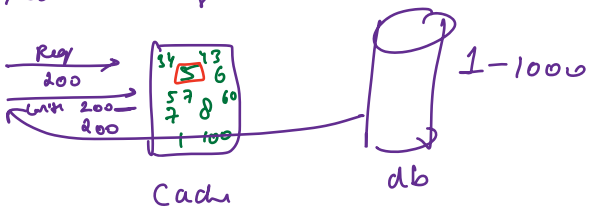
↓

fraud based.



Cache Eviction

Cache is limited in size



First in First Out

queue
simple

Least Recently Used

↓ time

↳ Hashmap + Heap
↳ Hashmap + DLL

If you see something, it is v. likely that you will see it again in near future.

Last in First Out

↳ whatever you've read recently will get removed.

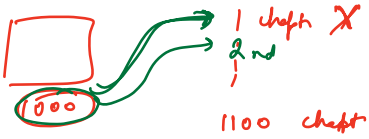
H/W → find out circumstances when this is useful.

Least Frequently Used

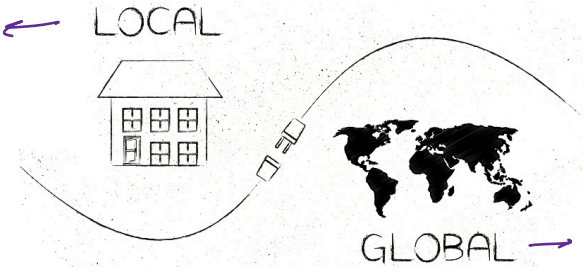
↳ evicted.

↓ read counts

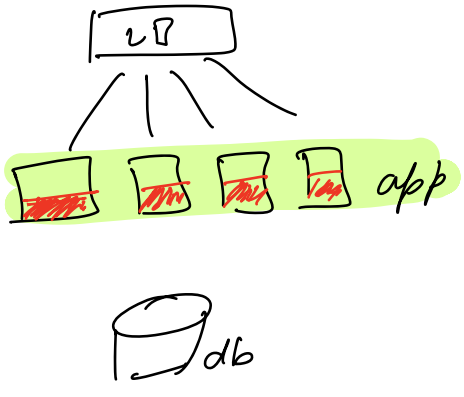
One Piece Manga Reading groups.

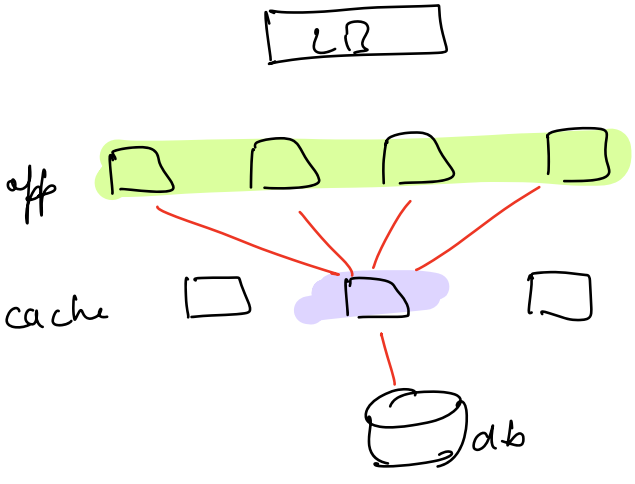


used by only 1 appserver. ←



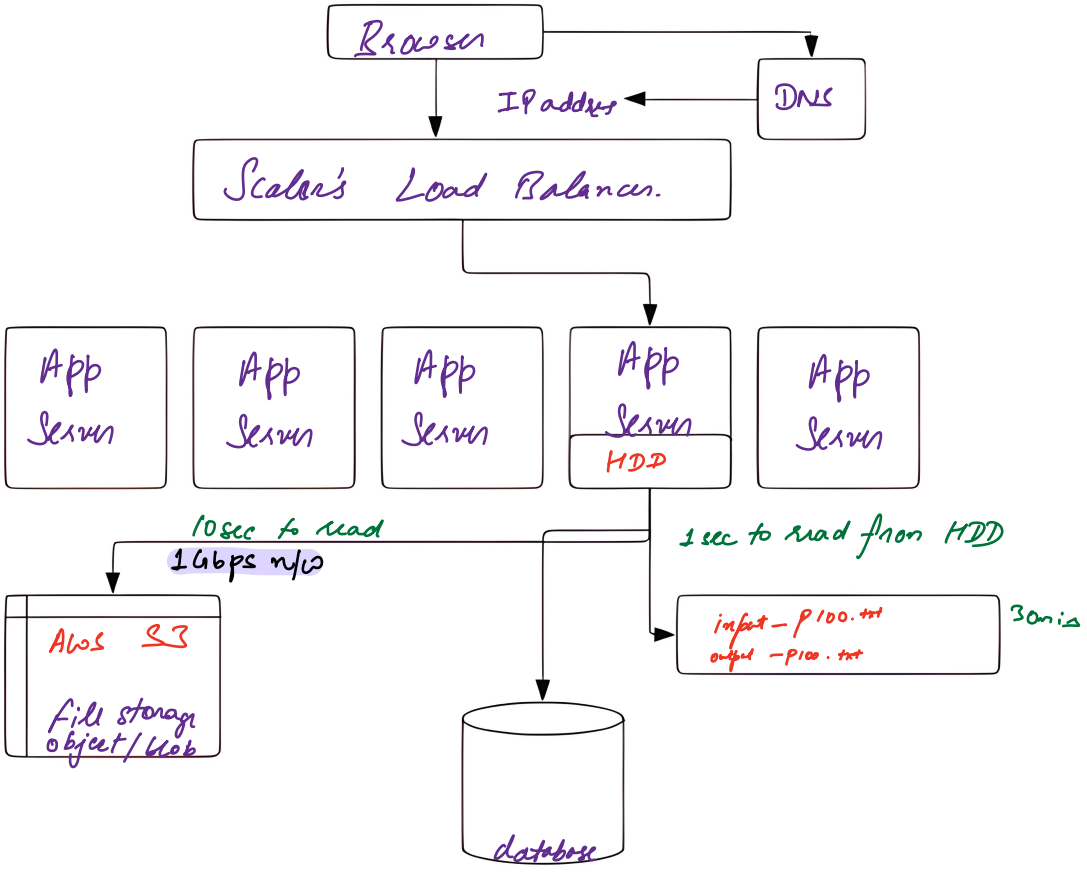
→ shared by all appservers





10:21 → 10:30

Online Code Judge



https://gist.github.com/jboner/2841832

? What does the AppServer need to evaluate a solution?

problem-id problem-metadata
user-id solved-already score
code language. input testcase & output testcase

given array → sort it
O(nlgn)
≈ 10^6 size
40-50 testcases

500MB input.
500MB output

? How large can this data be?

input, output } 500 MB each → 1 GB of testdata for a single problem.!

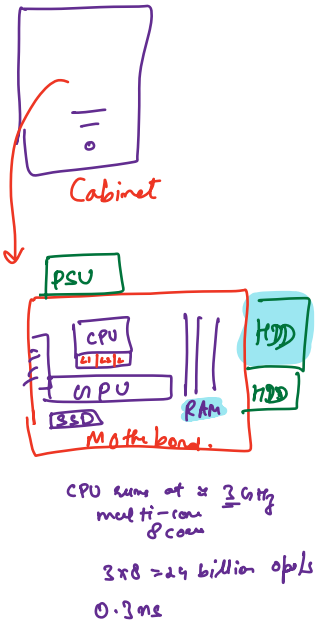
Change to the testcases

if a test case is changed → how do we update it in the local cache?

① at every write, push change to all appservers.

↳ write through cache

→ every app server has its own cache.
→ update every app server
→ wasteful



CPU runs at ≈ 3 GHz
multi-core
8 core

3 × 8 = 24 billion ops/s
0.3 ns

↳ write back cache ↳ TTL

some period during which data will be stale.

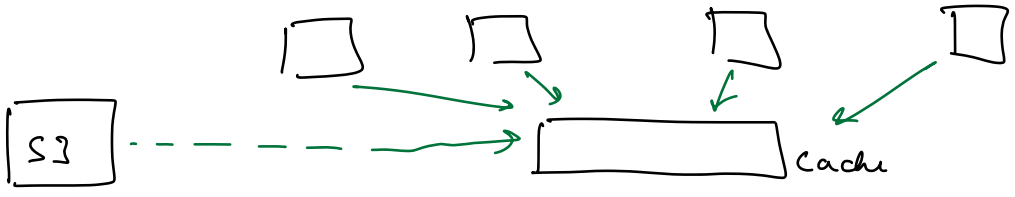
② TTL / write back cache.

bad solution → we want immediate effect

30 mins TTL → entire content gets over & testcases are not updated

30 sec TTL → most req will have to go to S3
no benefit of cache

③ Global Cache

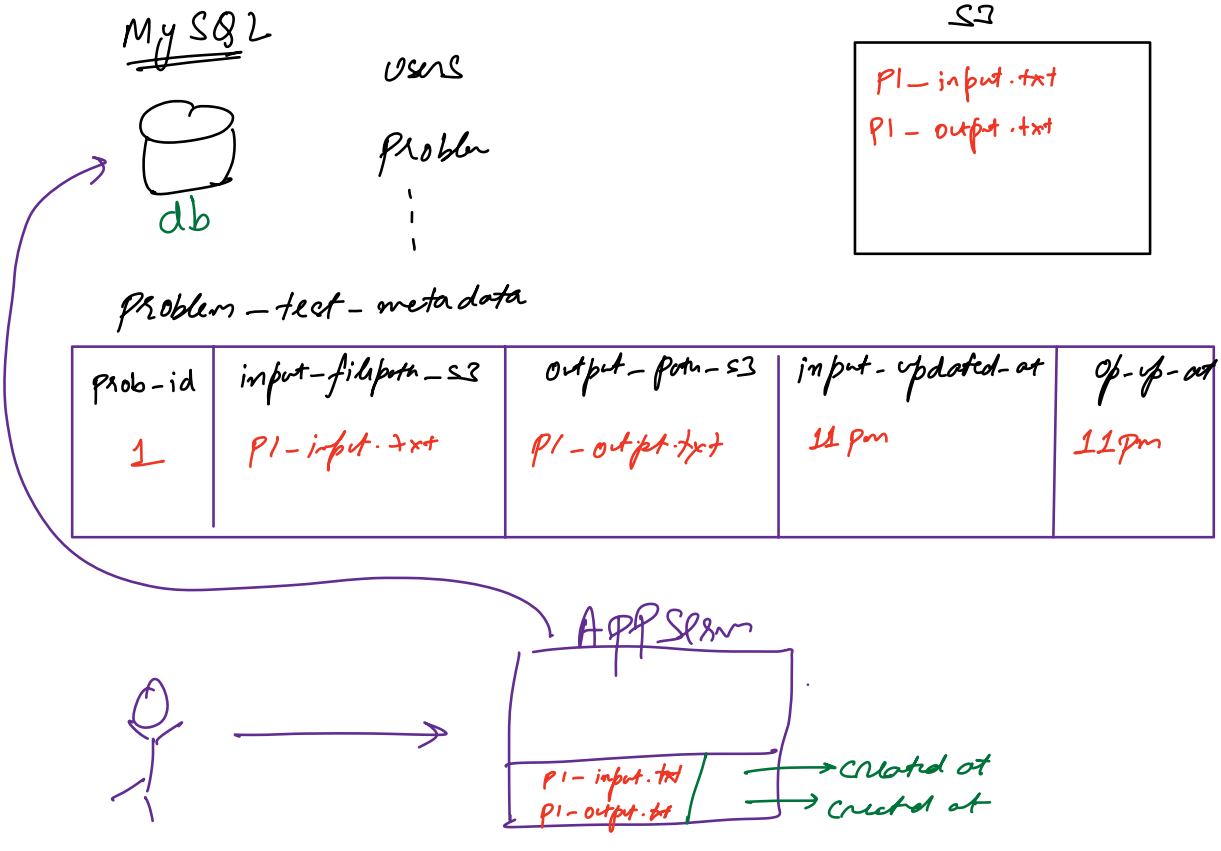


① alot of n/w traffic will come to cache

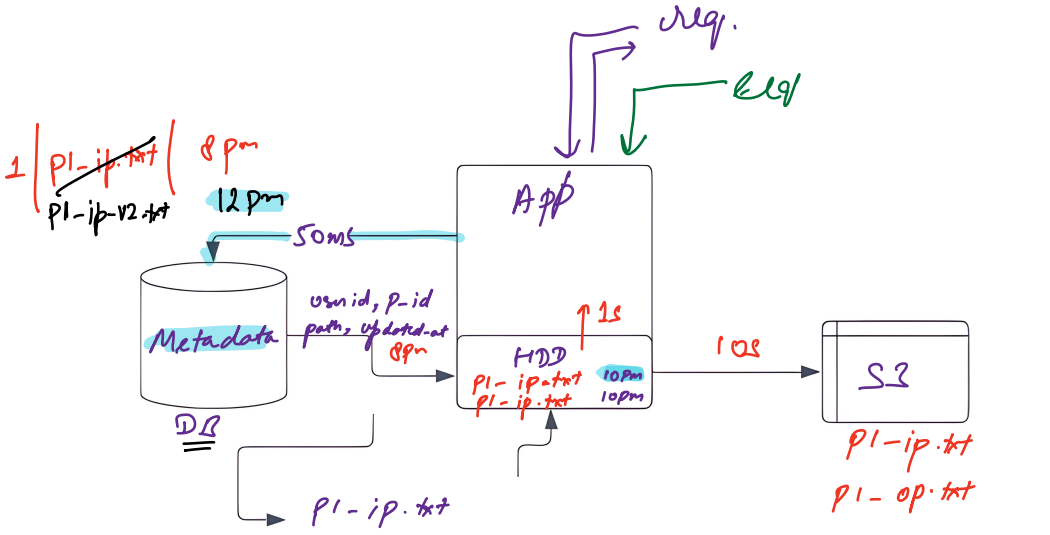
② if cache server has 32 GB RAM
↳ 32 files
a lot of cache misses.

③ store in HDD → 3 GBps SSD | ~300 MBps HDD

File Metadata in DB



Solution



p1-ip-v2.txt
p1-op-v2.txt

Test case → change

1. update file in S3
2. update db metadata

Contest Leaderboard

a lot of traffic
hundreds of submissions every minute

? How can we compute the ranklist?

user-id, name,
problem-id, score
user-problem
uid, prob.id, status (IE WA)

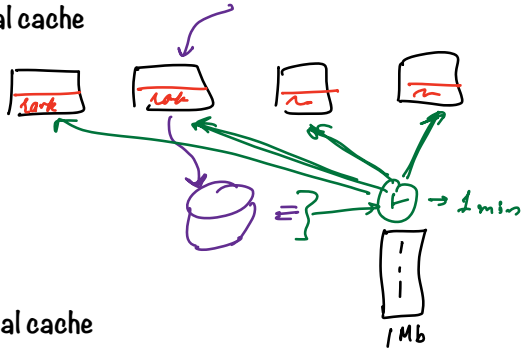
? How frequently does the ranklist change?

true ranklist updates after every submission. } several times per second

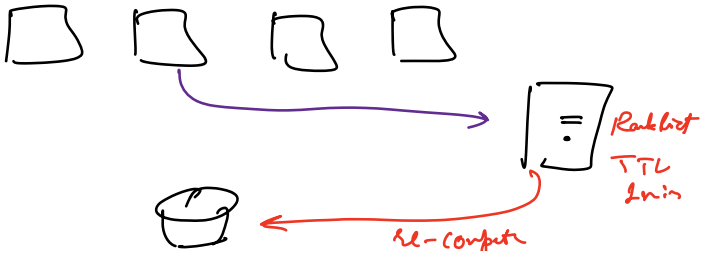
? Is immediate consistency important?

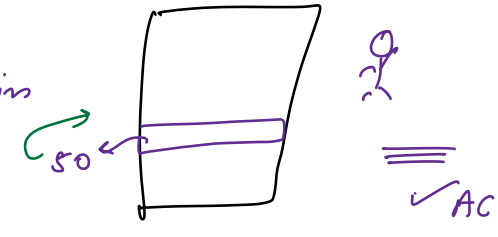
not important
ranklist updates every 1 min

Local cache



Global cache

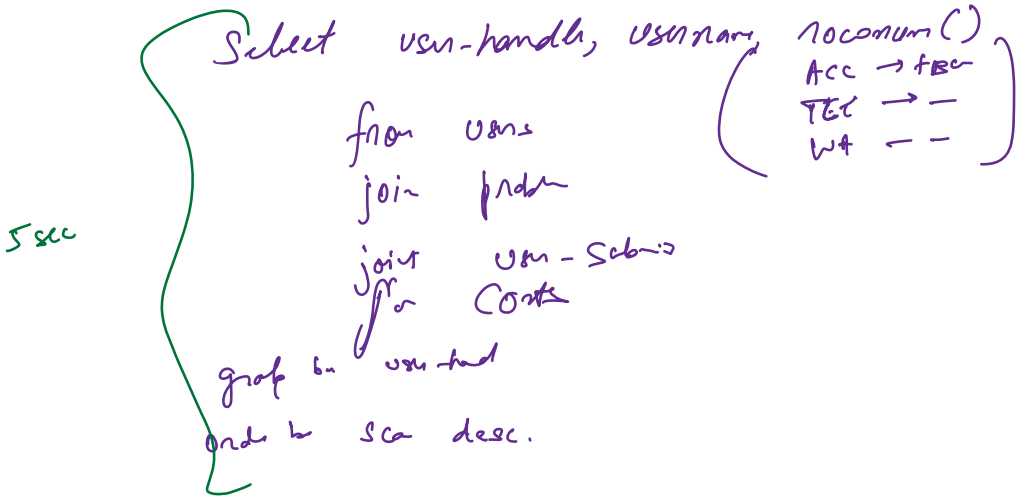




1 min

TTL/write around Cache
write-around (push)
re-computing ranklist once every minute
Not bad ✓

TTL (pull)
every 1 min each appserver will go to db & re-compute ranklist
100 times each minute
✗ bad



---

cronjob	global.
cron job that recomputes ever 1 min ✓	cron job recompute ever 1 min ✓
cache is local to each appserver	send data to global cache
send the entire ranklist to each appserver. 100 req/min	queries to read from cache are paginated → small data. ✓

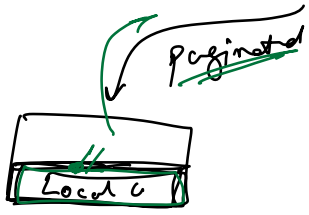
appserver can serve data from local cache ✓

user_handle → max 22 char string } 36 bytes
rank → 4 byte integer

100,000 users

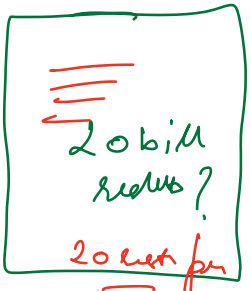
3,600,000 B

3.6 MB

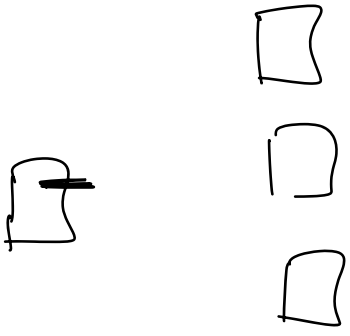


google

20 billion results fn in 0.1sec



10 cont



Redis

C

try.redis.io
redis.io/docs/data-types
university.redis.com

H/w

Singh-threaded key-value store (in-memory)
rich data types
String / Integers / Booleans
Lists / Sets / Sorted Sets

page = 100
20 results
per page

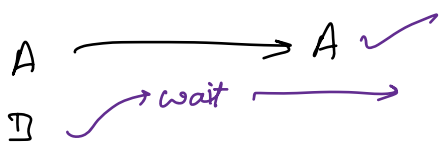
[2000
[1020

(1) give me rankList for page X

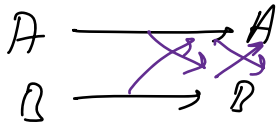
(2) given a user-handle, find rank

Key : Contest_id_ user_handle
Value : Score, Rank

Server is 1 threaded



multi-threaded



10:20 → 10:25

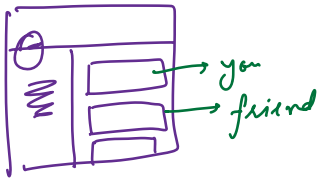
Facebook News Feed

? How can we compute the News Feed?

Posts made by me
Posts made by friends,
" " people / Pages you follow
" " groups
Posts by other that your friends have interacted with.

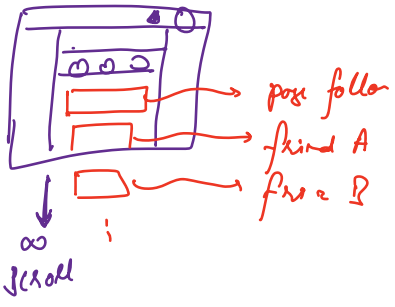
Recent data

Profile Page ✓




```
select * from posts
where user_id = ____
order by time desc
limit x offset y
```

News Feed ✗




```
select * from posts
join user
join user-friend on user_id = friend
join user
order by time desc
limit x offset y.
```

Posts

user_id	text	time	---

User

id	name	email	---

User_friends

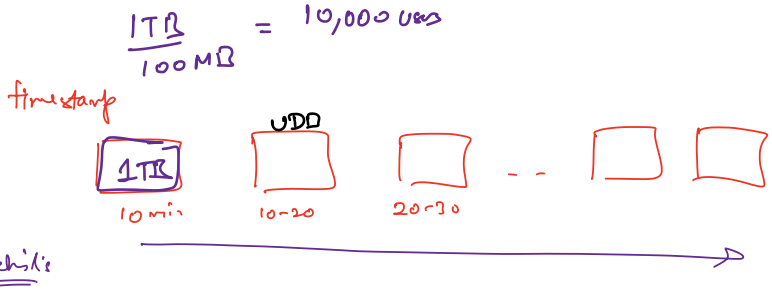
user_id	friend_id

? Does all the user data fit on a single machine?

No. shard data

? What is my sharding key?

post-id ~~X~~



Shard by user-id

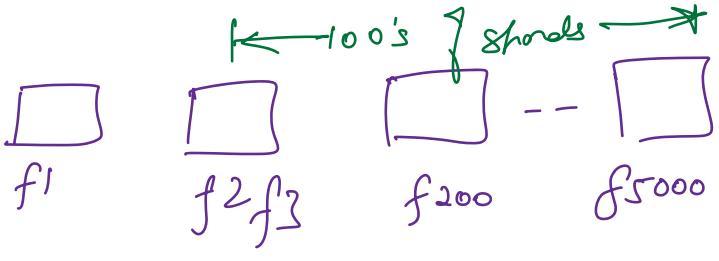
UDB
↳ user database

max size of 1 user's data

→ posting once every hour
→ all year for past 20 years

⇒ 12 × 365 × 20
<< 100,000 ⇒ 1KB ⇒ 100MB

? How do I retrive data for the news feed?



X not work

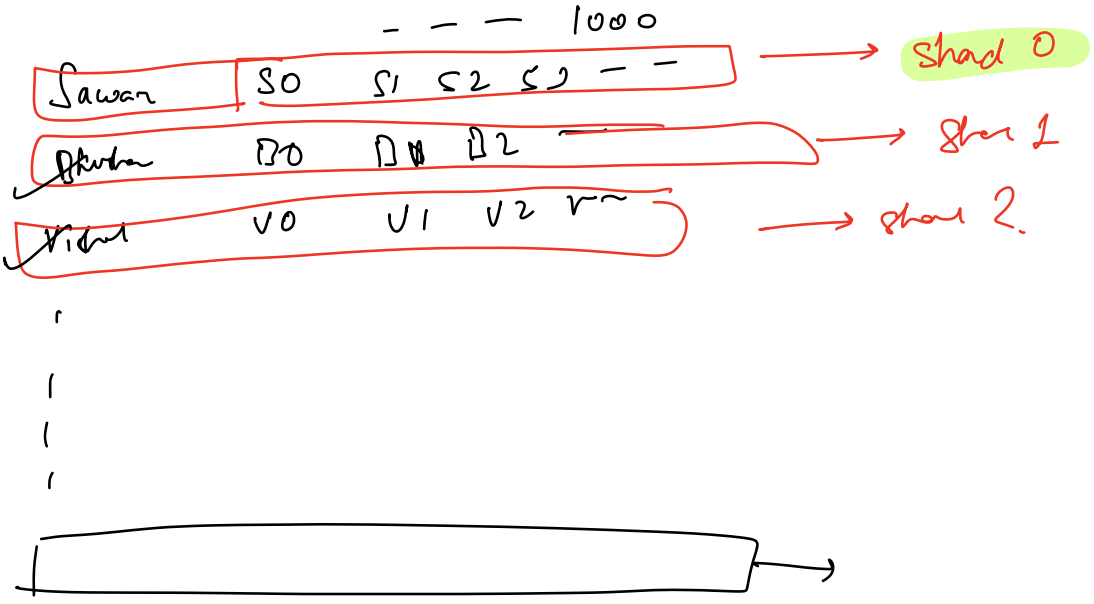
? How can you optimize the construction of the News Feed?

(1) Even if we cache it → first query is how to compute it

To compute a single news feed we need to hit 100 shards

expensive

1 → store posts in sharded dbs
2 → for news feed only store post-id


1000
Jawan
S0 S1 S2 S3
Shard 0
Shard 1
Shard 2.

Cache user -> news feed

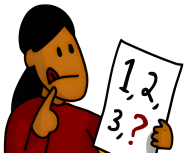
User id → [100, 2000, 3010, ...]

Or → [P1, P2, ...]

Sawan
Anurag
Nikhil

Bikramjit
P1

1. Billions of users, news feed has 100s of posts and each post is replicated in many feeds
2. Fan out updates: every post has to be copied to all feeds ✗
3. Update in news feed algorithm → v. hard



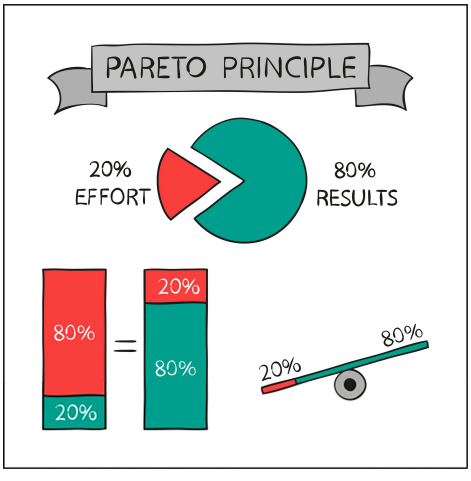
Back of the envelope estimates

Facebook → 1+ billion MAU

→ 800 Million DAU

posts made per day. ⇒ 800 M * 1% * 10 = 80 M posts

posts per day by avg user 10 posts (own estimate)



how many people would post on a day?

80:20:1 principle

- ↓ passive reading
- ↓ interacting (like, comments)
- ↓ generate content (posts)

? Daily post data

80M * 1KB = 80 GB

? Monthly post data

80*30 = 2400 GB

= 2.4 TB

Posts

P_id	u_id	content	Location	Media path	Timestamp
8B	8B	250	8B	250	8B

1 KB

4 byte → ≈ 4 billion values

Sharding → partitioning across slaves

↳ split data so that each partition has different data

↳ will improve read/write speeds → not the purpose

↳ true purpose → reduce data size

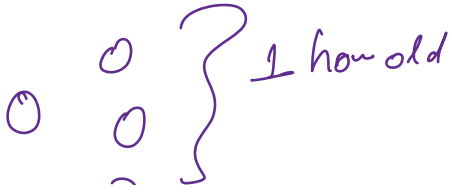
Replicate data → multiple servers having same copy of data

↳ improve read speeds

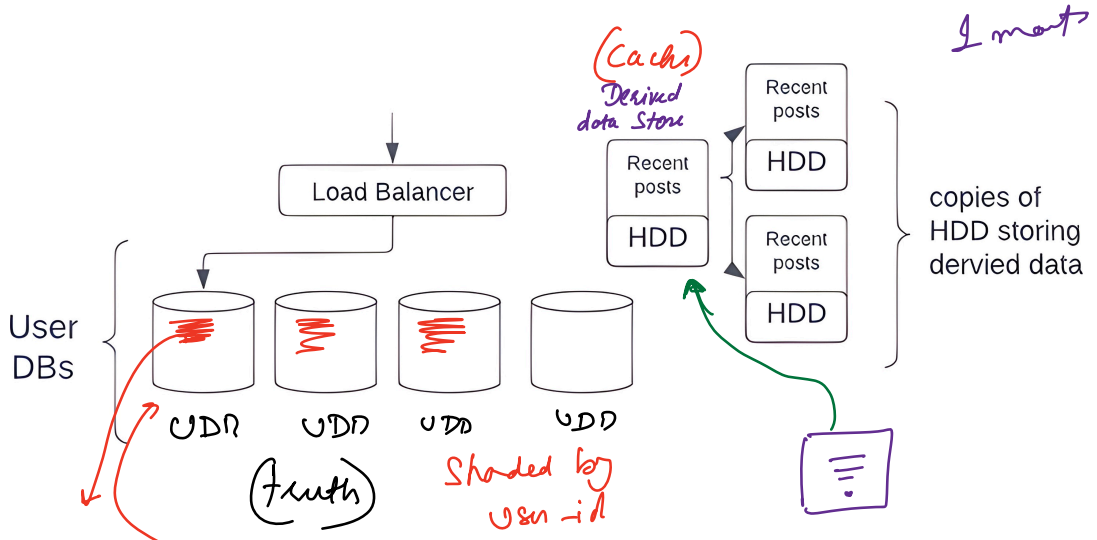
↳ availability

Ctrl + C

Ctrl + V



Single server !!



(Cache) Derived data Store

1 month

(truth)

Sharded by user-id

all posts for that user

20 year old posts & 1 day old posts

for each shard

```
Select *
from posts
where timestamp ≥ last 1 month
```

Akhil → news feed

1. fetch all friend ids of Akhil
2. select recent posts made by user friends.

```
Select * from
all_recent_posts
where user_id in ( ____ )
```

derived or cache

friends of Akhil

1. Update data ⇒ Cron for each db shard that updates last 30 min data into the cache
2. Remove old data. ⇒ every night at 6 am (least traffic time)

```
delete * from
all_recent_post
```

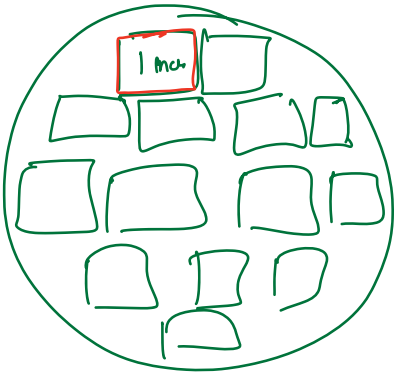
When timing > 1 min old.

CPU → Intel

clean rooms

ULSI → 3nm, 5nm, v.v.v. expensive

Ultra large scale Integration



Silicon wafer 1 million $

flip-flop → registers 50Mb

→ cheaper but slow

L1

→ cheaper but slow

L2

⋮

---

① write through → no stale data

② write around → cron job that updates cache using db

↳ TTL stale for some time

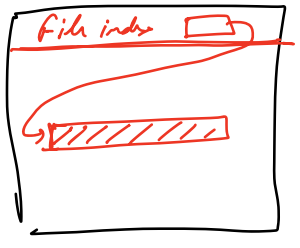
③ write back → inconsistent data

meta data → S3 → source of truth → did we have stale data

→ db metadata → No stale data!

S3 → File storage

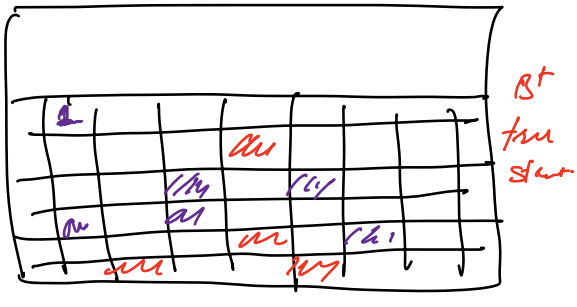
NTFS / EXT?



Continuous data access

DB

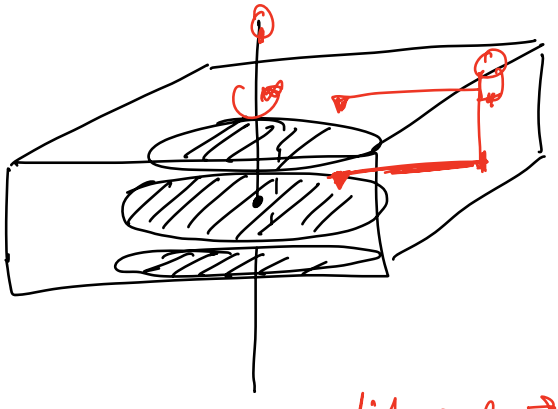
query data

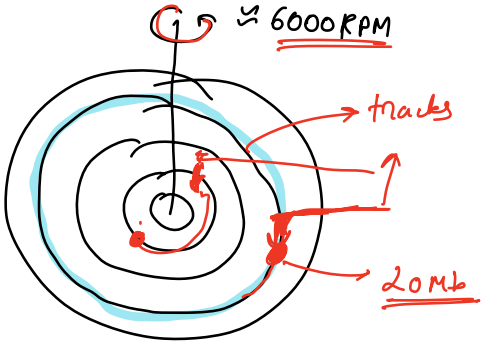


B+ tree

P_id | input 500n | out 500n

random data access





disk seek ⇒ 100ms

$\frac{6000}{60s}$ = 1 spin = 10ms

TTL → stale data

global → high n/w usage

↳ HDD → slow

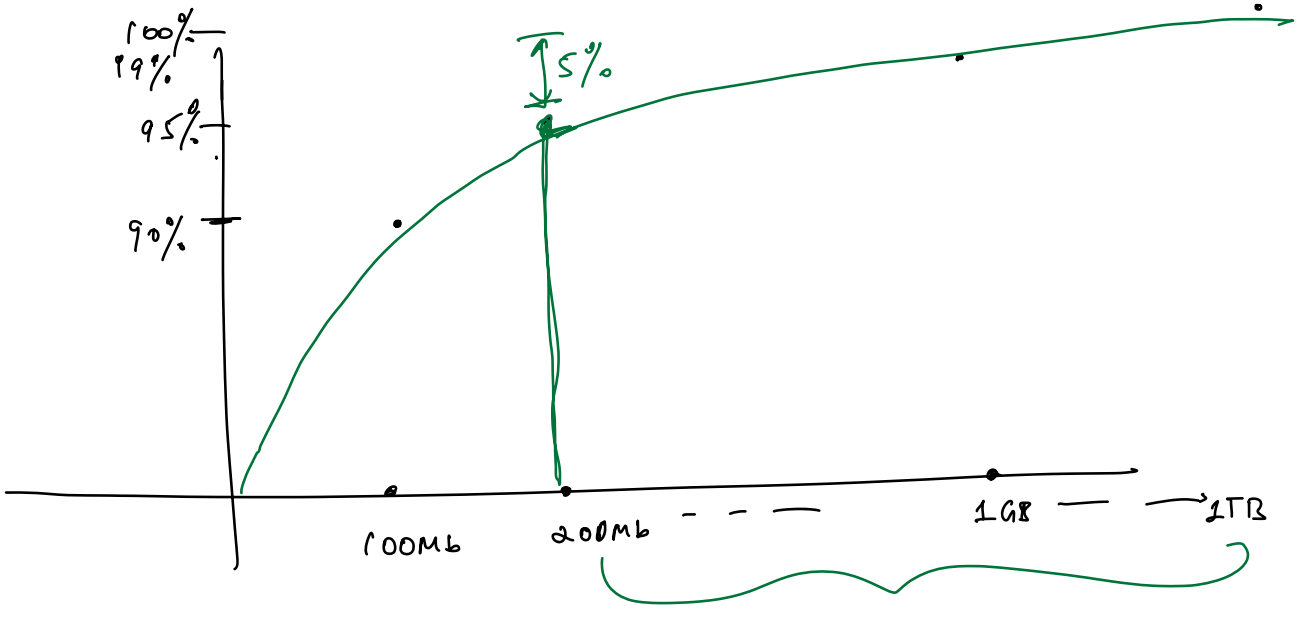
↳ RAM → small

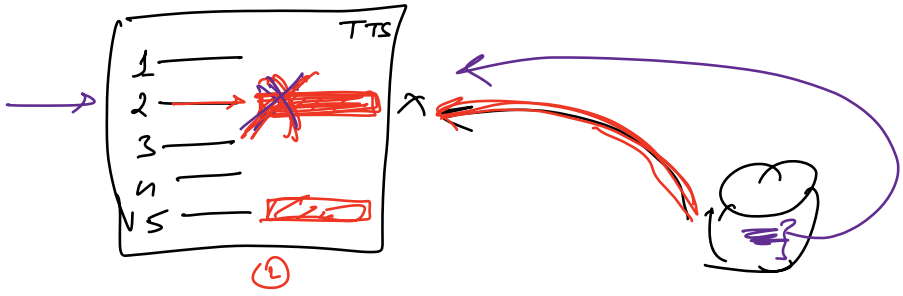
write back ✗ → inconsistent

write around → similar to TTL

DB → 1 TB

Cache → 100 Mb

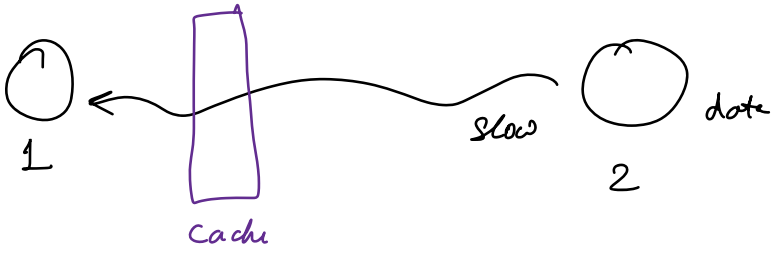


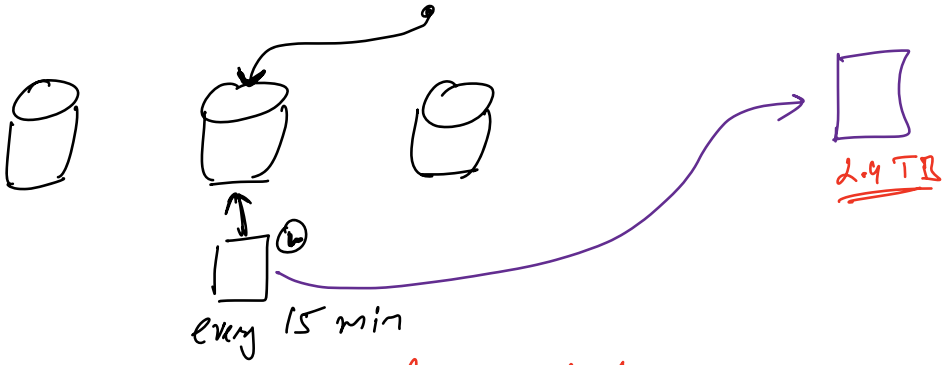


(1) wait for req (2) → expired → invalidate

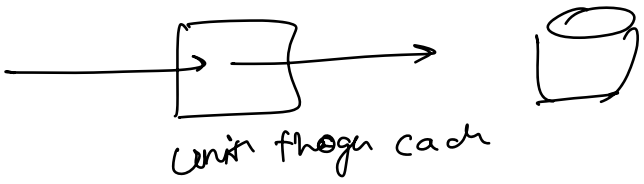
(2) Redis detect that (2) is stale → remove it

(3) cron → stale data → refetch fro db to update in cache





select * fro user-posts
where updated_at > last-15-min

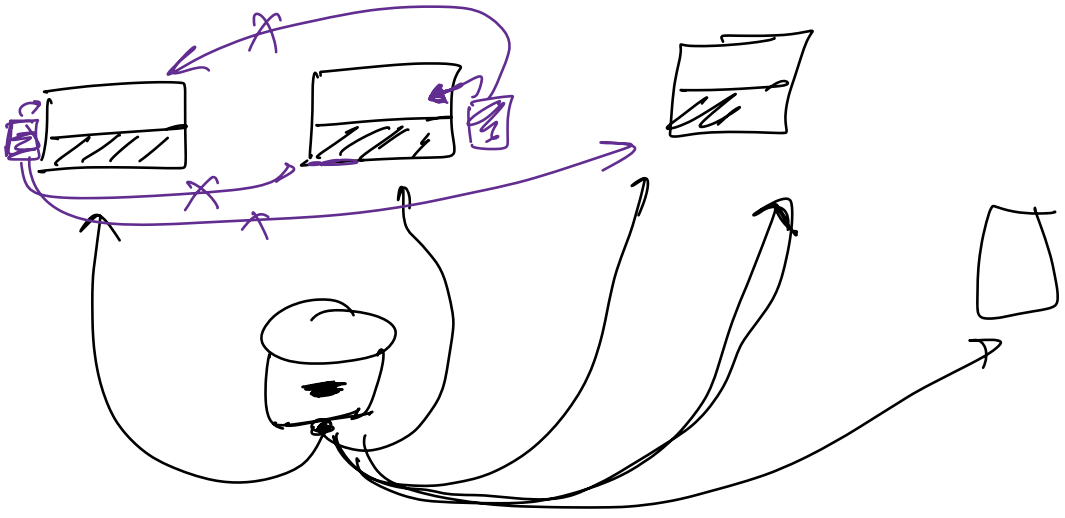


Write through cache

Cache & db must always be in sync.

TTL

Write around cache



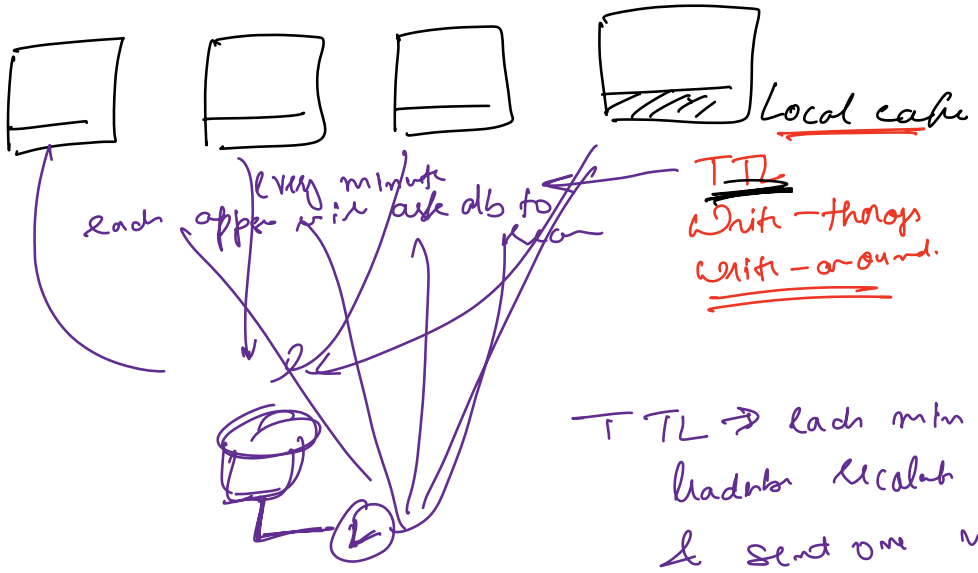
CDN → Cache for in-frequently changing data (mostly static) HTML/CSS/JS pdf txt audio/video/image

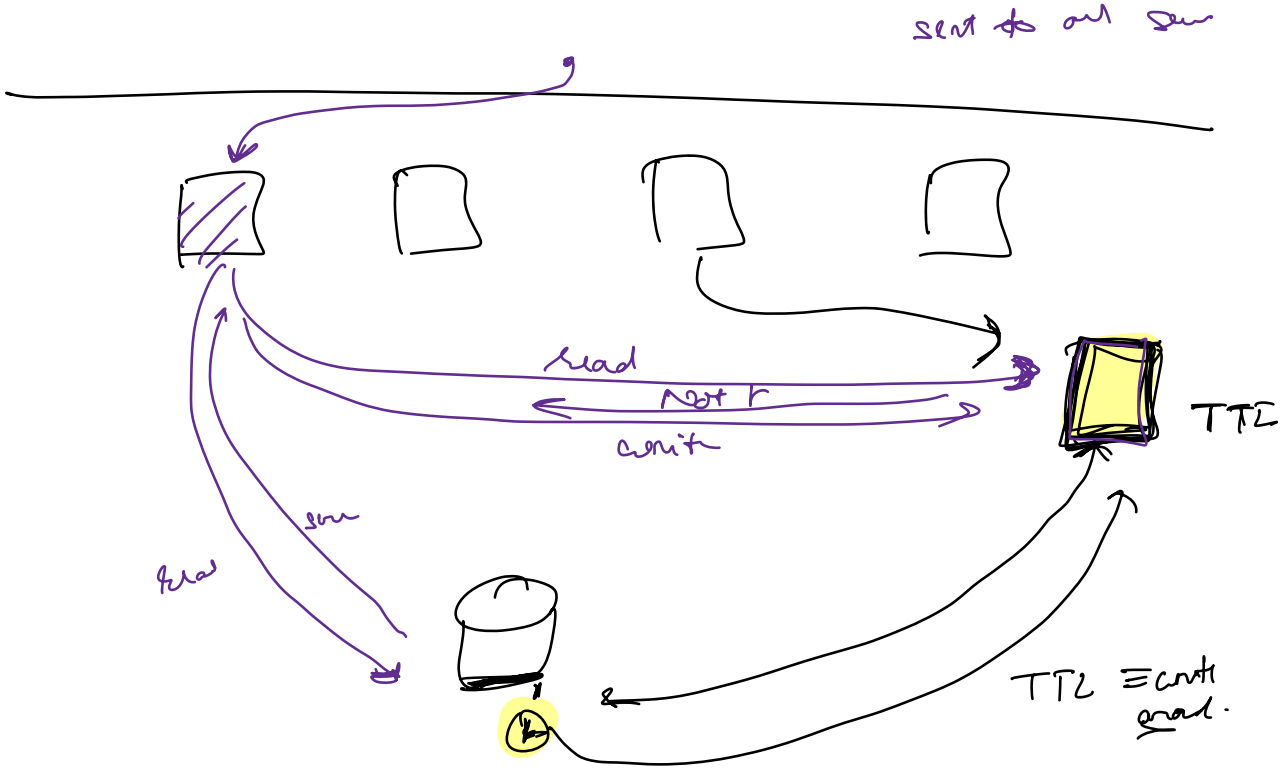
→ geo-located so that users don't have to hit servers far away.

S3 → source of truth to store files

db

Leaderboard.





1 Lakh req/sec

Consistency

fast responses

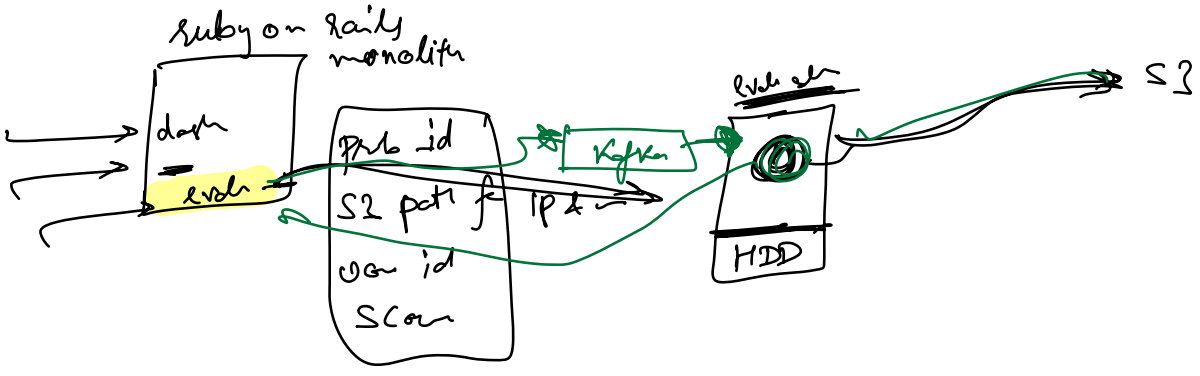
① serving | problem statement

② Read heavy / write heavy

Use case

input size → Any 10^6

sorting will work or not

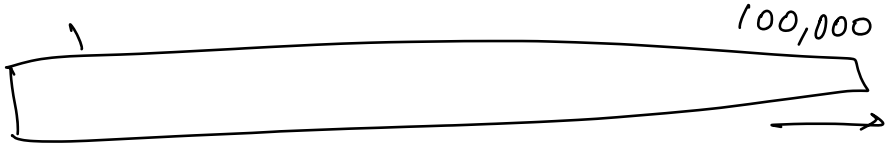


1 Billion 100 Billion post

User_Post_Views

↳ write back cache

18+ 50,000 visitors

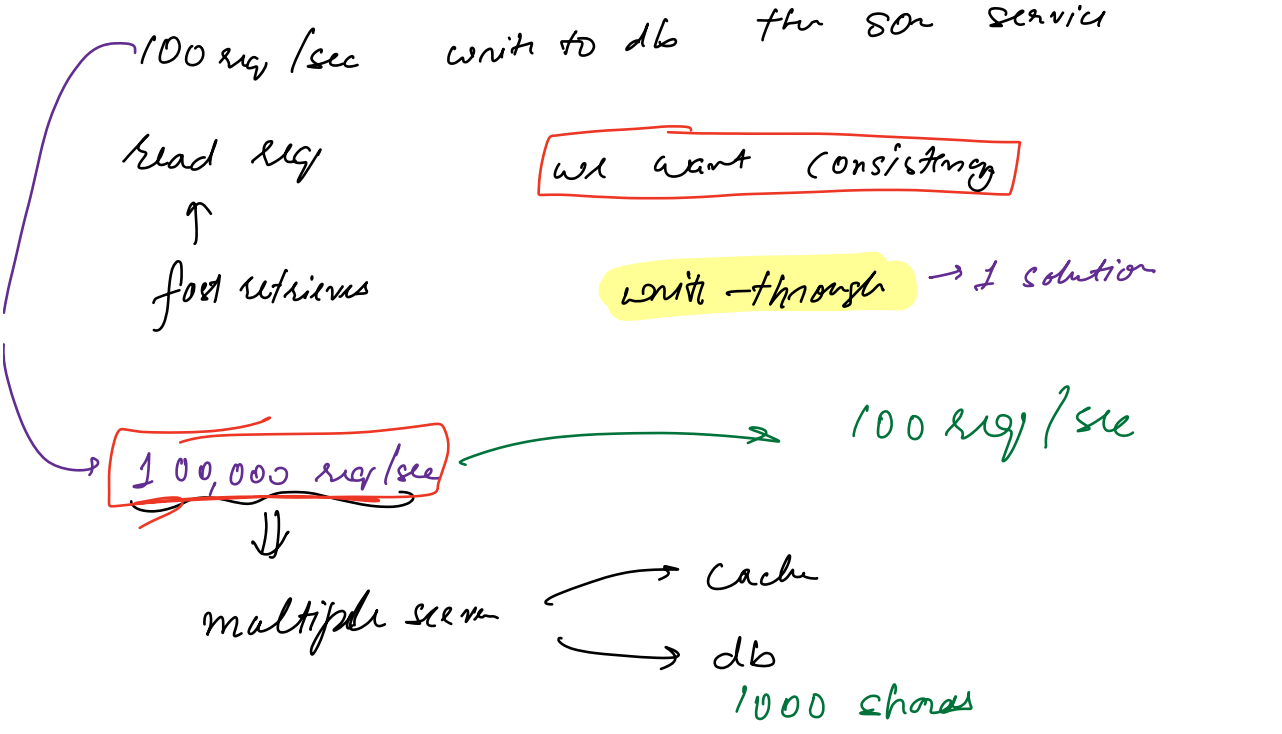


Page → 20 results

5322//20

↳ int div.

Range 5322



1.4 billion people

50% of the use UPI

0.7 B people

avg user will UPI how many day.

10 req per day

7B req per day.

$\frac{7 \times 10^9}{86400} = 1 \times 10^5$ req/sec

1+ Lakh per annum

25

Bachelors

Software

top 1% people of India.

# of users → MAU → 2 Billion
→ DAU → 800 Million

↳ activity is for one day

80:20:1 rule
80:20 rule

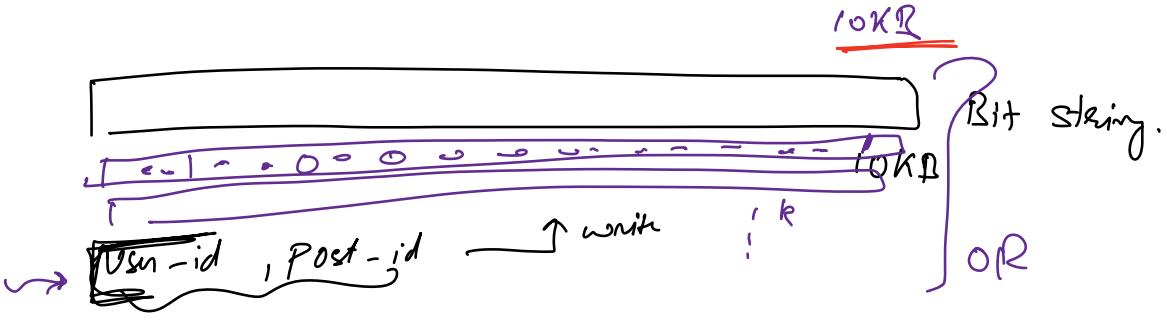
# of activities per day / 86400 → per second.

27,000 bookings per min — March 2020

27000 / 60 ≈ 450 req/sec

100,000 req/sec
insane!

S. Sudarshan → DBMS
Head of CSE → IIT B
→ DA of IRCTC



Hash(user-id, Post-id) --- to hash.


New value