Design a distributed cache

→ Caching
- motivation
- Types of caching

→ Distributed cache
- motivation

→ Strategies of caching
- read | write
- eviction

→ Design in OOP a distributed cache

Motivation

→ Browser
→ App

↳ Call your API

FE HAProxy

Round Robin
Base on load

Load Balance

Microservices

monolith

JMT
IO
JMT
3
JMT
JS
JS
JM
3
Scaling

100

1 X A

1 X B

→ Create new section

→ Create a larger classroom

Scaling

↳ vertical scaling

→ increase capacity of the machie

→ 100 new 200

→ 200 → 1

300 → 1

↳ horizontal scaling

↳ creating more classroom

↳ 100

→ more changes

→ load →

→ disadvantage

→ expensive

→ distributed

— 1 machine

100 GB — 10 GB

vertical

+10

Diagonal

+1. micro

+1

+3

n

n3

→ shard

→ replicate

Access Pattern

→ which data is being loaded very frequent

→ which API is called a lot

→ query is taking most time

---

1 Million

- 80% of GET USER

→ replication | sharding

→ access time

→ Faster at fetching data

persistence

→ no

→ disk backed

→ locality of reference

T2 - 1

T3 - 0

T 2000 - 4

Read | Write Strategies

→ Cache-aside

application server

- gets the data from the DB
- and stores in the Cache
- stone data

cache aside

App server

- cache invalidation
- eviction
- Multiple data stores

- DB

Cache - Global

read-through

Separation of concerns

Latencies

aside

→ through vs

→ smart vs dumb cache

→ latency → request has to be with him on the DB

Banks

Can we write to the cache later

Can the cache store the data in DB later

Kacher

write-back

latency

consistency

---

backrus through

bank, stock

consistent

consistent

→ eventual consistent

5:58 | 6:03

→ cache-aside
→ r/w through
→ r/w back

→ Distributed

→ consistency back

Eviction

→ OOD design fun cache

Inconsistent

Sticky thread sessions

- Read Your Own Write
- Consitent Hashing

Cache - 10MB

Cache - 11MB

eviction policies

LRU - Least Recently used

LFU - Least frequently used

- frequency

1 - lol min →

2 - loo[illegible] lo

3 - 1000 [illegible]

4 - 1000 min

10000

Data - 100 GB

Distributed

Sharding

Divide our data

each cache will be queried for a hit

extensible

latency

latency t

ex tensible

→ Large amount of data

→ Region

→ Number of request

IM — Icache

— Shared your data

— replicate cache

Class Diagram of a Distribute cache

→ Requirements
- clarifications
→ Distributed Cache
- support multiple write
- support multiple eviction policies
- low latency
- partitioned / sharded

{

"Key":{"Value", created, updated, accessAt

"Key1": "Value1",

LFU

MRU

Field

set(K,V){

// Setting the data

data.set(K,V)

// Update the metadata

metadata.update

}

getKey

|| Dataset

|| Metadata Set

y

→ notified when a change → perform a metadata update

Association
is-a
has-a