Api Rate Limiter

Monday, March 25, 2024 8:53 PM

FUNCTIONAL REQUIREMENT

1. Limit no of requests for an entity and blocks requests after threshold
2. provide API rate limiting across diff services.

NON FUNCTIONAL REQUIREMENTS.

1. Availability over consistency (one or more requests going forward is okay)
2. Reliability
3. Speed (low latency)

TYPES OF THROTLING.

Hard throttling : Strict limit.

(3) limit.

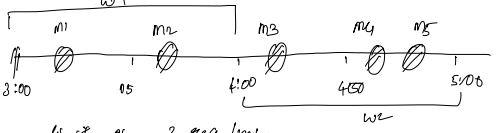
Soft throttling : 500 reqs/hr + 10% exceed limit

550

Elastic throttling : allow additional requests if system has resources

ALGORITHMS:

1) FIXED WINDOW ALGORITHM

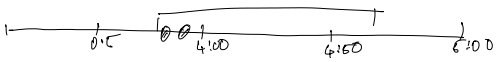


limit of 2 req/min

w1 — m1, m2 are served.
w2 — m3, m4 are served
m5 rejected.

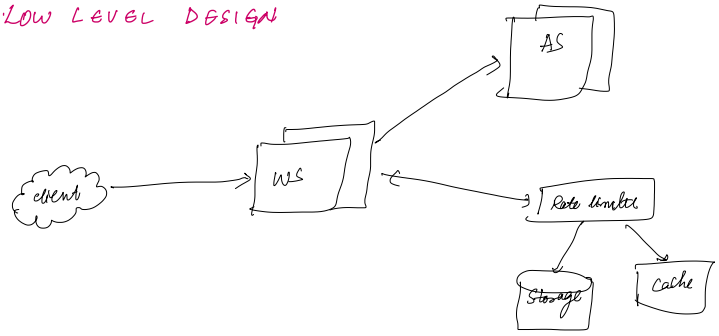
If 2 reqs are from second half of and the other from 1st half.

2) SLIDING WINDOW ALGORITHM



window shifts every time a request come in.

LOW LEVEL DESIGN



1) FIXED WINDOW

Hashmap

normalize it to a minute.

key : value.

"userId" : {counter, epoch} (or start time) object

we can delete the record once the minute is changed.

500 reqs/hr.

Names : sliding window

Ex: 10 req/hr

3 req/min

userId : 8bytes
Epoch : 4bytes
Counter : 2bytes
500 reqs/hr.
20 bytes (redis cache)

minutes

8 + (4+2+20) x 60

8 + 1560 + 20 = 1588

for 1million users 15

SHARDING AND PARTIT
[userId]

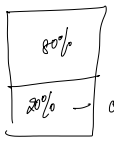
18 is not

company | cafe

MAC address — unique

Cache Eviction : LRU

Types of rate limiters.
1. Request rate limit
2. Concurrent rate lim
3. Fleet load shedder



critical tasks

4. Work utilization

1) Critical
2) POST
3) GET
4) Testing.

example : { "user 14308" : { 1, 1427400843 } } assume : 3 reqs/min.

memory requirements:

userId : 8 bytes
count : 2 bytes
epoch : 4 bytes (32 bit number)

Total : (8+2+4) + 18 (bytes) ⇒ 32 bytes
overhead of hashmap

1 million users active.

32 bytes × 1 × 10^6 ⇒ 32 MB.

SLIDING WINDOW

HashMap <key, or DLL
{ userId : sorted set < ? > } 3 reqs/min are allowed.

user 14308 14308742
14308744
14308748
14308750

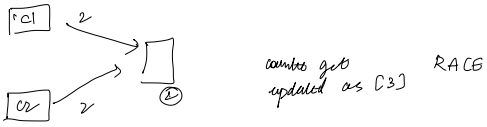
{ user,14308 : [14308742, 14308747, 14308748]
14308774
if the above 1 min.

14308774

userId : 8 bytes
epoch : 4 bytes
500 req/hr
20 bytes overhead sorted set.

8 + (4+20) × 500 + 20 = 8 + 12000 + 20
= 12028 bytes
overhead of hashmap
= 12 kb.

1 million users — 12kb × 1×10^6 ⇒ 12 GB



counter get updated as [3] RACE

LOCKING MECHANISM

IN memory cache - redis

redis lock

Trade off for both is hybrid

sliding window with the help of counters.