Agenda.

↳ Design Rate Limiter.

⇒ ChatGPT : 35 requests | 3hrs.

⇒ LinkedIn : Limit on no. of connections request we can send|accept per day

⇒ Twitter : View 10K tweets|day.

Rate Limiter.

↳ Prevent DDOS attack

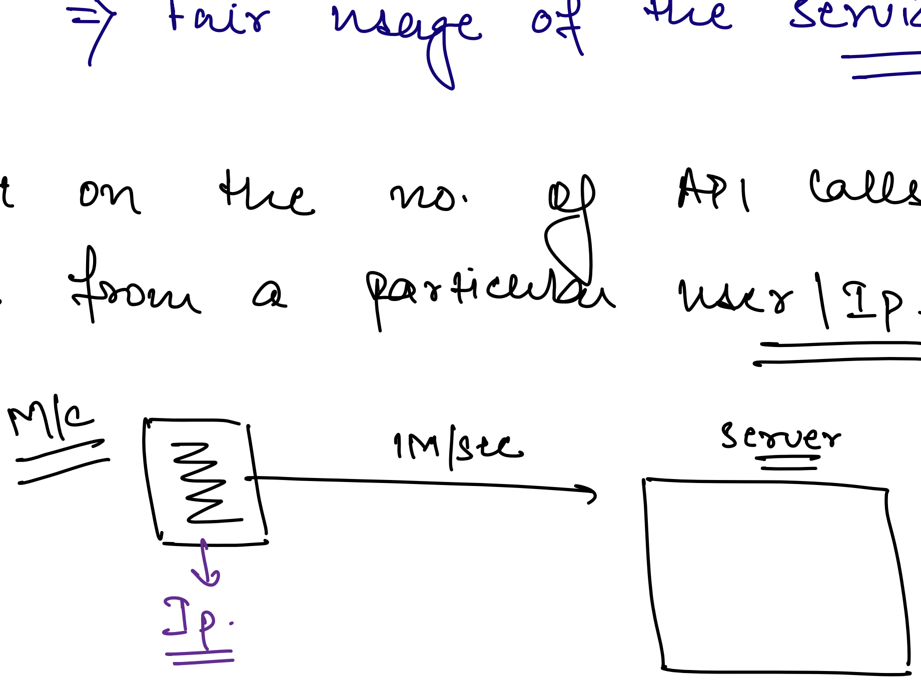
↓

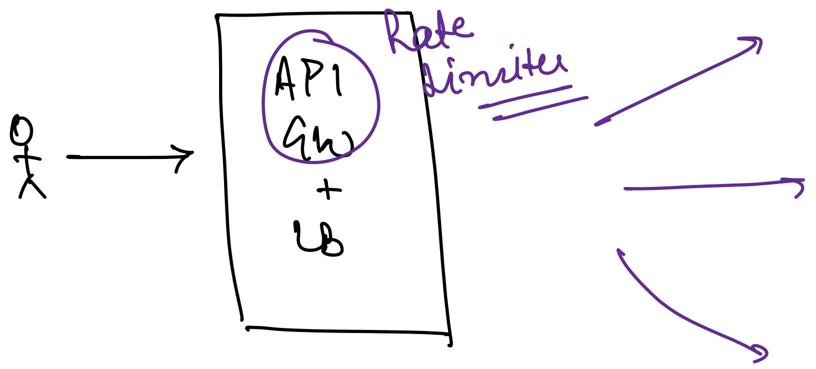
Distributed Denial of Service.

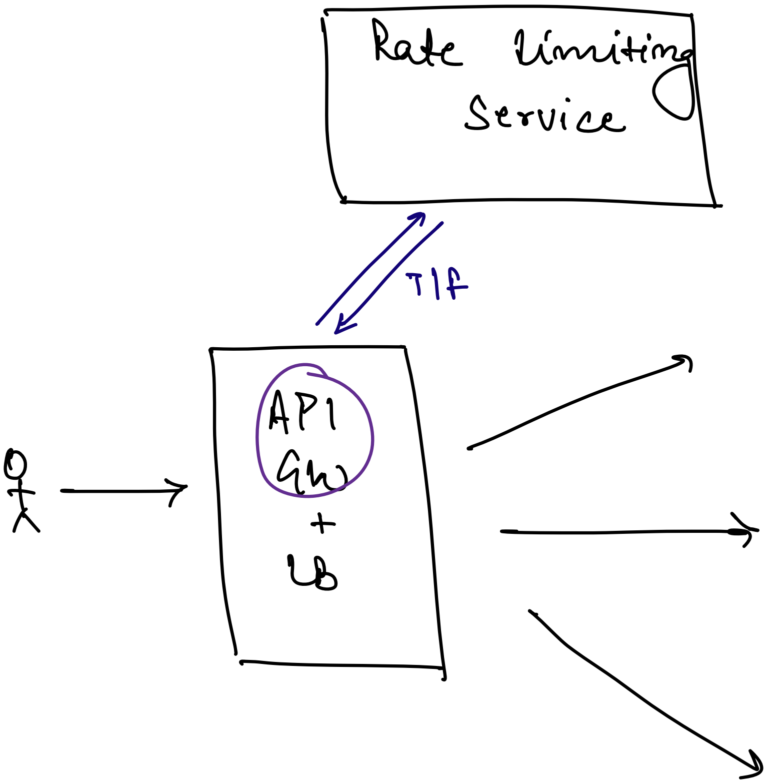
⇒ Our system can down.

⇒ fair usage of the services.

⇒ Limit on the no. of API calls can be made from a particular user|IP.







⇒ Http Status Code.

- 429
- Too many request

Functional Requirements (features)

→ Ability to limit the requests based on a particular criteria.

→ Configurable.

→ Different API's can have different limiting criterias.

API endpoint : 10 requests | user | min.
/createPost

/viewPost : 10K requests | user | Day.

Non functional Requirements

(Trade Offs | Architecture)

⇒ Consistency vs Availability ✓

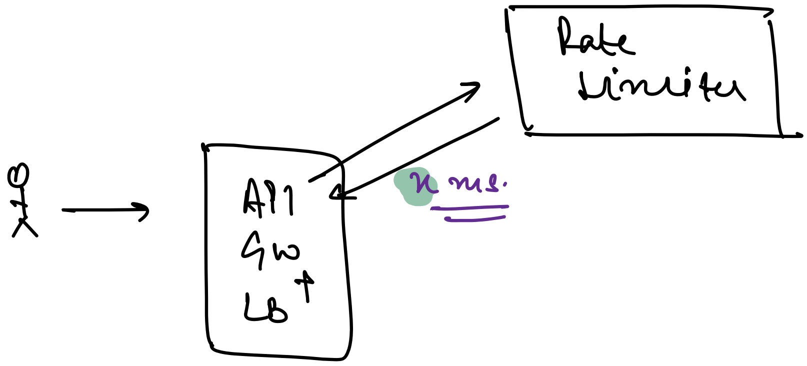
CAP.

High Availability.

⇒ If Rate limiting service is Unavailable then our entire service will be Unavailable. So, High availability is MOST of Rate limiter.

⇒ Consistency (vs) low latency.

⇒ Super low latency.



⇒ Rate limiter is going to be used for API call, and if its taking more time then latency of all other API will increase.

Back of the Envelope Calculations.

Twitter | facebook newsfeed.

- CreatePost (≡)
- ViewPost(≡)

⇒ 3B Total users.

DAU : 1B

Avg No. of view post Api Call | user | Day = 50

No. of Api calls | Day (View post) = 50B.

No. of create post API | day = (10% of 1B) * 5

= 100M × 5

= 500M

= 0.5B.

Total # of API requests = 60B | Day.

$$= \frac{60 \times 10^9}{10^5} \text{ qps.}$$

86400 sec | Day → 10^5

$= 60 \times 10^4$ qps.

= 600k qps

Read Heavy or Write Heavy.

⇒ Both Read & Write Heavy.

Storage.

userId/IP	API id	Count
1042	1xy2	100

8b   4B.   8B. → 20B

api_endpoints.

	id

1 Billion users × 10 APIs × 20 Bytes.

$200 \times 10^9 \times$ Bytes.

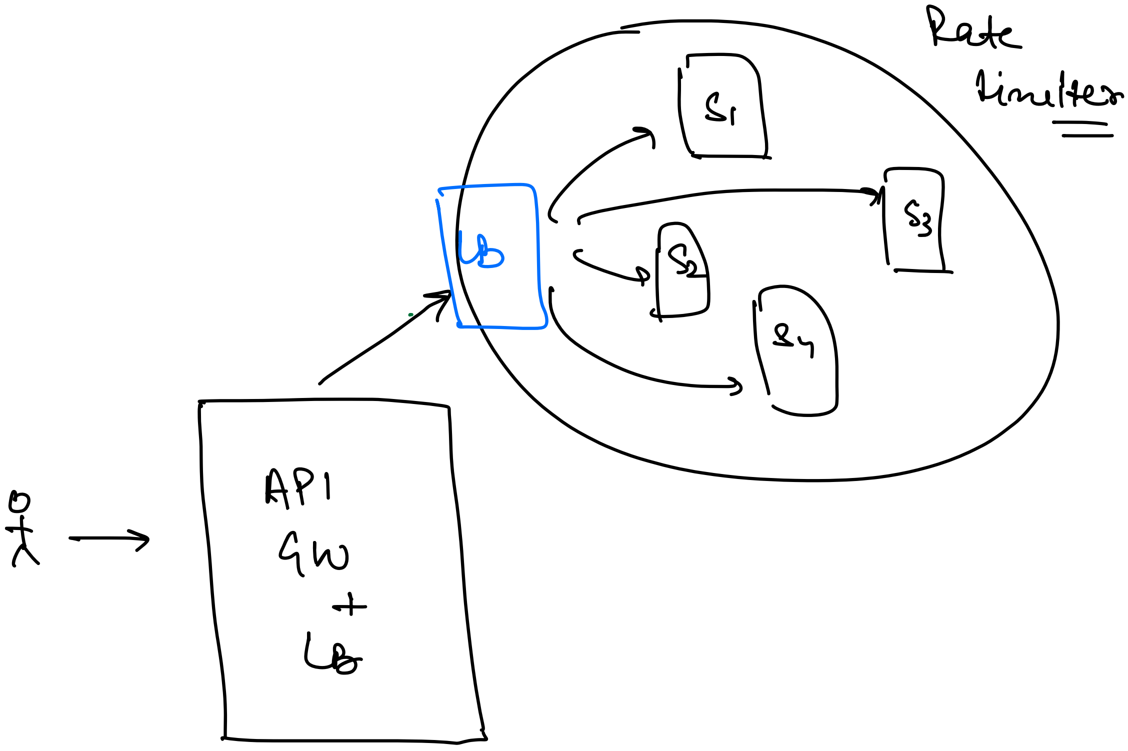
$$\boxed{200\,GB}$$

⇒ SHARDING ✗

⇒ No need to persist Data.

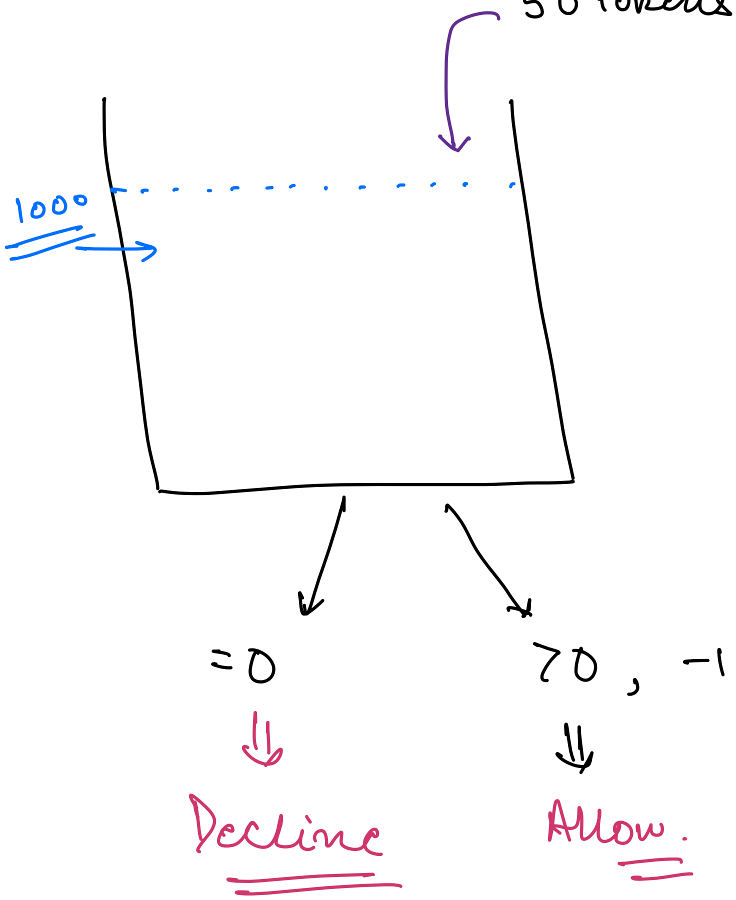
⇒ Cache ✓

⇒ Distributed Rate Limiter



1. Token Bucket Algorithm.

50 tokens/min] Refil rate.



==userid== = 1041

==API== ⇒ /acceptConnection

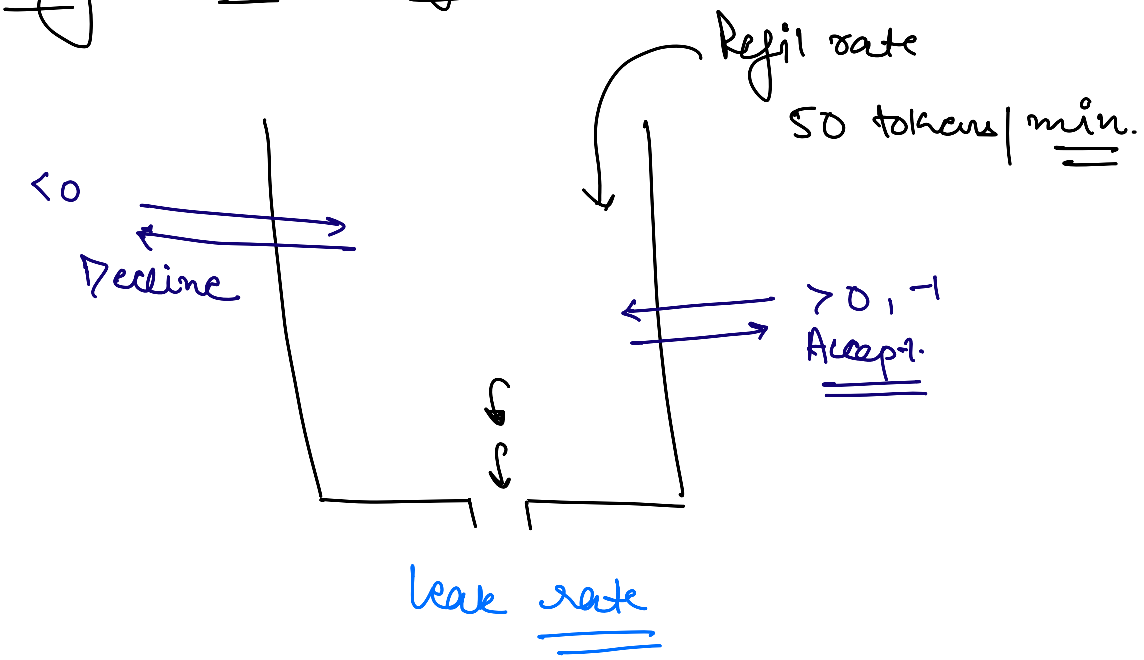
⇓

50 req/min.

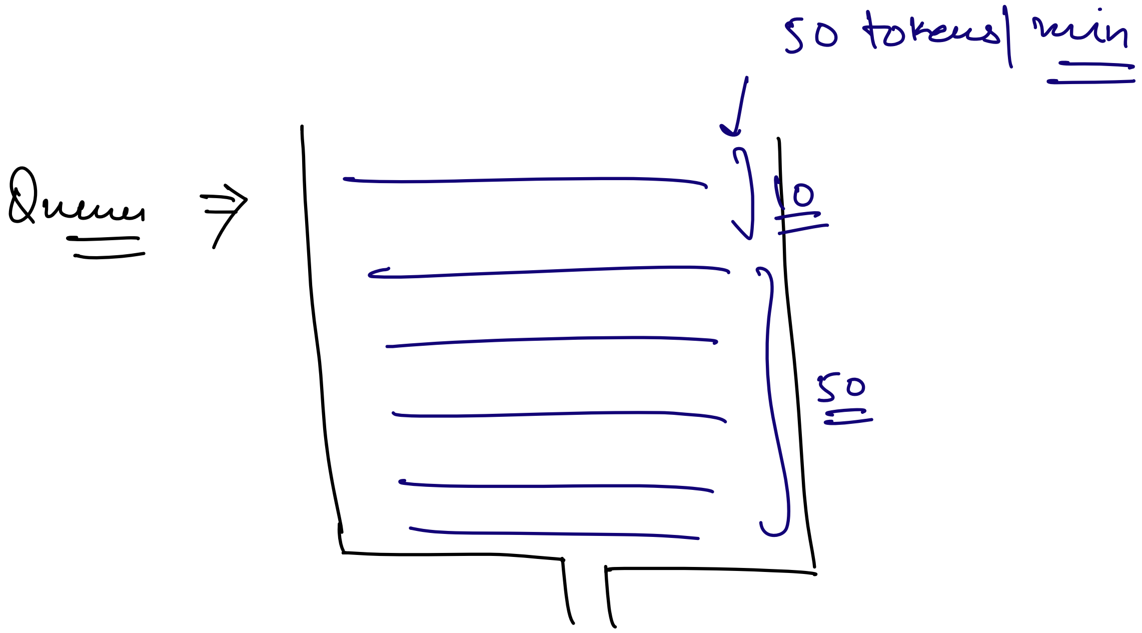
userid + API ⇒ Count.

⇒ Bursty Traffic.

2 Leaky Bucket Algo



3.



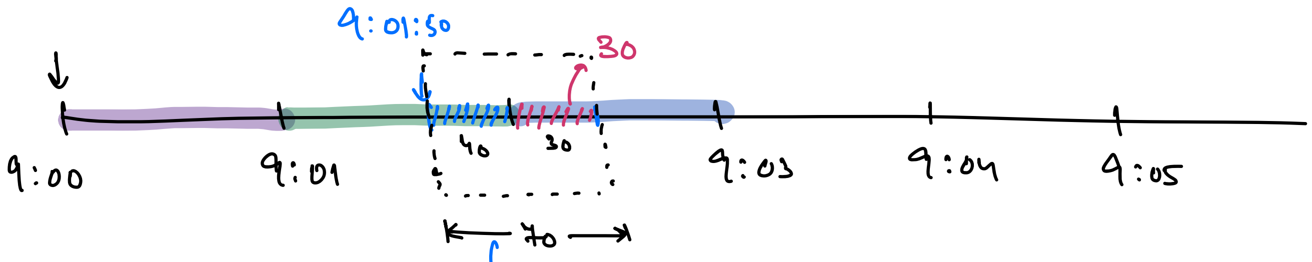
⇒ Resource extensive algo.

⇒ Unfair to the new requests as the older requests will get executed first.

4. Fixed Window Counter

50 requests | min.

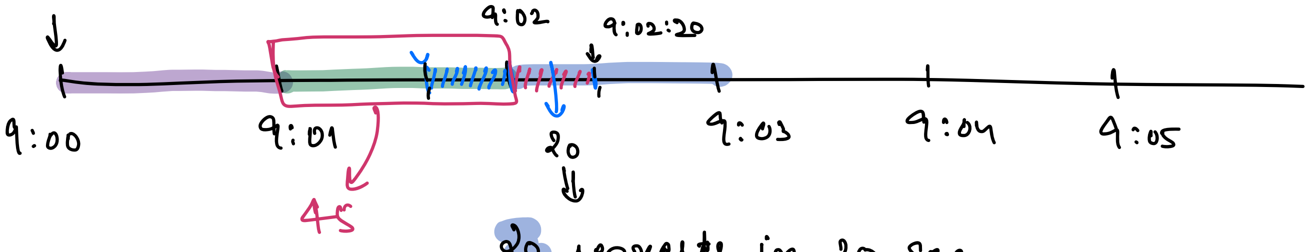
At the start of every window, reset the counter.



peak traffic

5. Sliding Window Counter

Fixed Window Counter + Solve peak traffic



20 requests in 20 sec.

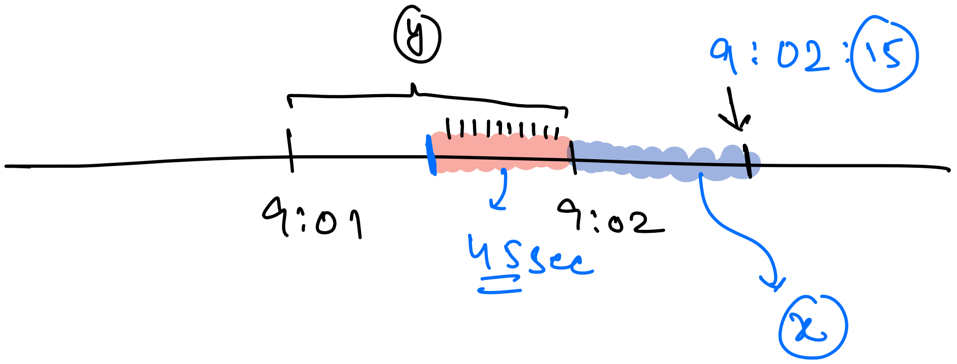
60 sec → 45 req

40 sec → $\frac{40}{\cancel{60}_{4}} \times \cancel{45}^{3}$ $\boxed{30}$

# of requests in the current min

$$= 20 + \frac{40}{60} \times 45$$

$$= \underline{\underline{50}}$$



# of req $= x + \frac{\overset{3}{\cancel{45}}}{\underset{4}{\cancel{60}}} \times y$

⇒ Approximation.