Rate Limiter

Thursday, February 19, 2026 6:22 PM

Rate Limiter

Brief - It handles the req from client. It just like the traffic controller for the backend infra
- it allows the defined number of requests to the infra for the specified interval of time

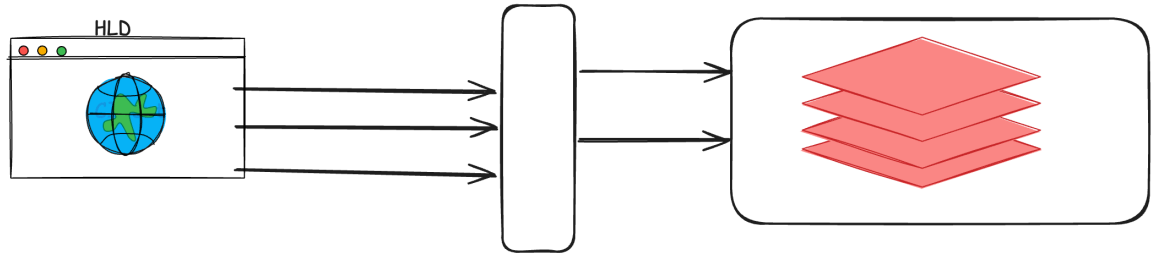
Bouncer Metaphor
- club has the capacity (backend infra)
- People are waiting in queue(request)
- Bouncer - only 5 people can enter to the club in every 2 min(job of rate li

Benefits
- it protects the infra from dDos attacks

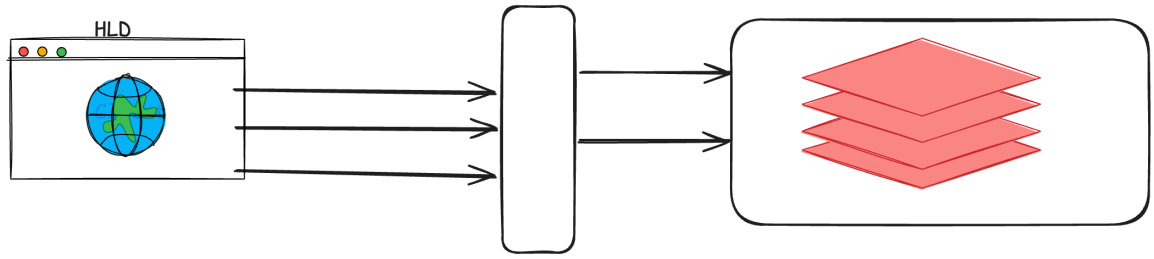
Functional Requiremnt
- Limit Req - drop req, if exceeds thresold
- id - Rate limit based on ip, userid o api key
-Feedback - Return a clear error e.g 429 - too many req

NFR
- Low Latency - it should not slow down the actual api call. Either Yes or NO in 10 ^-6 sec
- High Availability - if it is down , it should not crash the backend
- Distributed Accuracy - the limit should be global . e.g in a cluster of 10 servers, 100 req/sec
  should be global not per server.
- Memory efficient - It should store as less data as possible (e.g counter and timestamp)

HLD



Client



Client

The Core Algo
it can use any one from the below based on the business requirement

1. Token Bucket -
    - Bucket holds N token. Each req take one. Bucket will be filled at const rate
        - e.g. if a user can make 2 posts/day , 20 comments/day , add 150 frnds/day , here user needs 3 different buckets based on the maxtoken size for each bucket
        - e.g if we are allowing 100 req for ip , then each ip address needs a bucket
    - Pro - Allows burst of traffic
    - con - two variables are in the system
        - bucket size
        - refill rate

        So it could be tough to tune

2 . Leaking Bucket -
- Same as Token buket , req enters the bucekt and processed at fixed rate
    - FIFO queue. If q is full, req drops else added

- Pro - Memory efficient , as q size is limited
    - due t fixed rate processing, it is needed where fixed rate is req
- Con - Bursty traffic drops immidiately
    - two params - bucket size and q , so tune could not be easy

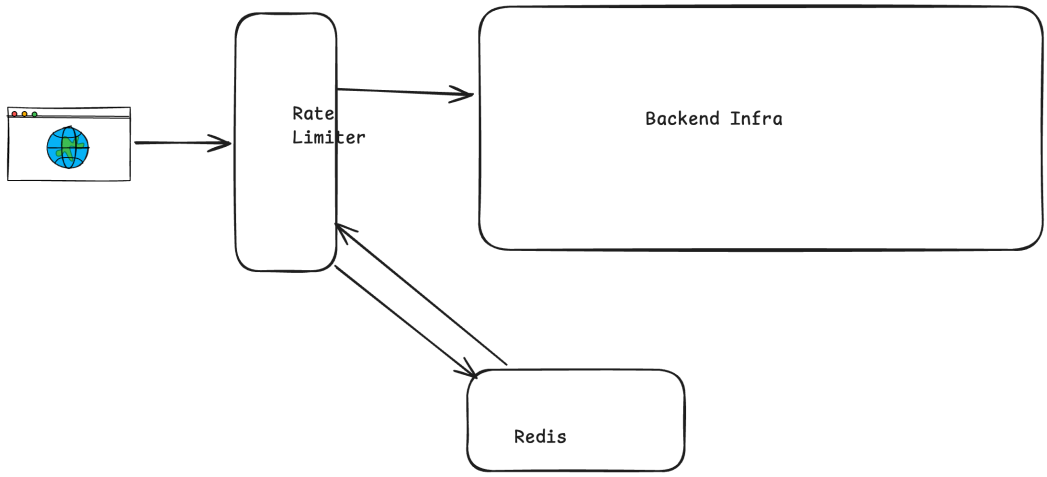
3. Fixed Window -
- Counts req is fixed timeframe
- pro - easy to implement
- con - for the spike, it could allow more than the allowed number

4. Sliding Window log -
- It fixes the issue of fixed window

pro - good accuracy
con - consume memory , as timestamps are being logged in the Redis even for the rejected req

5. Sliding Window counter -
- its a hybrid fixed window+Weighted avg
req # = curr +(% of the prev window*total no in the prev window)

pro - very effiecient. handles traffic spike
Con - as it is an approximation of the prev window, so wont work effieciently if the req is strict look back



- 429 - if a req is rate limited , then the http code it returns

Challenges in Distributed Env
- Race Condition -
- if both req acts on the Redis to get the counter and increment the val by one, so it will be counter+1 however the
correct val would be counter+2.
- Solution - Lock - But it will slow down the system.
- better soln - Lua Script
- sorted set DS in Redis
- Synchronization issue
- in distributed arch , multiple clients and multiple rate limiter. As the web tier is Stateless, if a client will
send request to different rate limitier could cause problem
- Solution - Sticky session - but it is not scalble nor flexible.
- Best soln - Use a centralized data store like Redis

- Best soln - Use a centralized data store like Redis

Cnsider Latency
- it is best to set up multi DC Ratelimiter. Edge location is preferable

Observability -
- Check Rate limiter algo is efficient
- if it failed to serve the traffic spike which are business critical , then need to reconsider the rules

Some suggestion on the client side
- use client cache to avoid being rate limited
- understand the limit and be polite on sending req