**API rate limiting Design**

API rate limiting is also known as throttling. API throttling allow you to control the way API is used.

Throttling allows you to set the permissions as to whether certain API calls are valid or not.

Rate limiting types: user based, ip-level, concurrent connections., etc.

Tiers of Throttling:

API-Level throttling

Application-level throttling

User-level throttling

Account-level throttling

Flow chart for API Rate limit: -

API Rate Limit Counter Counter

Cache

Application

Web Server

LB

Cache

Application

DB

FW

* User-level or Account level throttling via token bucket algorithm:

Here each user/account will have some token access whenever there is request, counter will be decreased using API Rate Limit Counter.

Same information will be shared across multiple nodes using distributed cache like Hazel cast or Redis.

API rate limit counter system is centralized system which keeps track of the tokens based on account/user.

In the above diagram, each node contains application along with distributed cache servers which keeps monitor the API rate limit counter.

First application checks at local cache server instead of centralized server.

This counter will be reset whenever the configured time is over like 1hour/ 1 min.

For example: there is a request at 10:00:01 and configured time period is 1 min

Counter will be decreased when there is a request for specific user till 10:00:59. Whenever it reaches to 11:00:00, the counter will be reset.

Database: Using relation database (mysql). Because we need to use the relations between no.of requests and user/account.

Api\_rate\_limit\_config - This is to configure the userIds/accountIds with threshold limit

Api\_rate\_limit\_date – This is capture the latest counter values and date.

Challenges that may face with above approach is **Inconsistency:**

1) Inconsistency could be solved by using sticky session, where one user session requests will be routed to same sever always but this leads to heavy load on server.

2) Another approach is to use the locks on Counter system but it’s a performance hit.

3) Counter values may be erased when system restarts.

**Throttling approaches:**

1) Relaxing rate limit: Here we can consider little inconsistency if there is no problem.

Example: If system set to 10 requests per minute. Due to inconsistency, it may allow 11 requests or 12 requests.

2) Hard rate limit: Here requests should be throttled when it reaches configured value. To achieve the same, we should use locks on services.

To overcome #3 problem:

We can add life cycle call backs (application listeners) – before server going to restart/shutdown mode, update the latest counter value in database. When server coming back to server the requests, update the counter system with db data.

Till server comes to live, rate limit will be processed against database. So that we can achieve highly available system.

* IP based throttling can be achieved using rules at firewall/webserver level. This throttling required mainly required to mitigate DDOS attacks.