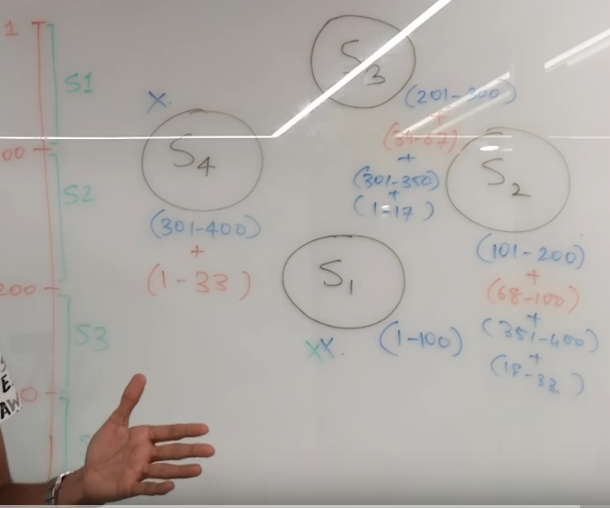
# Distributed Rate Limiting

## Problems

### Cascading Failures

Let’s start with we have a multiple server load balanced and each server is handling a range of request as in below diagram in blue text.



**In summary:**

S1 -> (1 – 100)

S2 -> (101 – 200)

S3 -> (301 – 400)

S4 -> (401 – 500)

Now, in sudden S1 failed, so the loads of S1 got distributed between others highlighted in red.

Say S4 was completely busy and loaded with its initial range and getting added parts of request of S1, S4 now super loaded and crashed after some time.

Now load of S4 also divided between S3 and S2 and now S3 is performing 200% more load and possibly could fail and so as S2. Which is cascading failure.

### Going viral/ Black Friday

These could be the days when the sales are very high and number of request is huge

### Job scheduling

If we have a cron job and which got scheduled to send some message to millions of users in a single time. Ex. Christmas message on 25th of Dec 12:00:00

### Popular post

If a popular person publish something and notification got sent to everyone in a batch processing way, but once everyone at a same time hits the pay to see/video to load

## Solutions

### Cascading Failure solution

This failure is a race against time, the time gap/delta required to bring back the new server till the other server distribute the failed server load.

**Solutions:**

* Stop serving load for the failed node, because serving some users is better than serving non users.
* Every server can have a **queue** with every servers compute capacity and once the queue filled with the compute capacity request count (ex. 300 req/s) for further request we are going to ignore the request with “Try after sometime” message.

### Going viral/ Black Friday

We can go with **Auto scaling** or **rate limiting.**

**Note:**

With auto scaling if we are pay as we use and some external agent or api is making humongous request to our system (Brute force attack) then our environment could auto scale to adopt the request and we will be losing money for paying for use.

### Job scheduling

Job should broke into chunks and each chunk should run in a time and after that next should run.

### Popular post

There could be some information about the content which is metadata of that and users has very less to do with that.

In a youtube video, the number of views, likes, dislikes, comments are metadata and those people barely care and we can provide some approximation of number for say for the views. It also could be done in a smarter way by analyzing the data of his previous post on how views changes based on time.

So **Approximate/statics** could be a solution here.

### Caching

Caching the common data rather every time hitting the data source could improve the response turnaround time.

### Pre-scaling

With an approach to assumption of future failure could have extra compute ready with us. Like if we need 4 servers to process this much request we could have 2 more in standby.

### Coupling \*

Sometimes coupling can be an option for faster performance, where we store the data which we get from other system in our system.

Ex. UserService is taking to Authentication Service (External service) to get user name and token. Where if we cache username and token in our system if username hasn’t changed since one hour in past so we can avoid one network call here.

But again based on the risk factor this kind of decision or design can be carried out.