Tiny URL Generator System Design

1) Functional Requirements:

a) Generate Tiny URL for a given long URL

b) Fetch long URL when Tiny URL is accessed

2) Non-Functional Requirements:

a) Low Latency

b) High Availability.

3) Estimations:

a) Length of the Tiny URL

Say x requests per second and we need to store these URLs for 10 years

Therefore.

Total NO. of URLS $= x * 60 * 60 * 24 * 365 * 10 = y$

Base 62:

[a-z] [A-Z] [0-9]

26 + 26 + 10 = 62

If length = 1 $\Rightarrow$ we can have 62 URLs

" = 2 = 62^2

" = 3 = 62^3

:

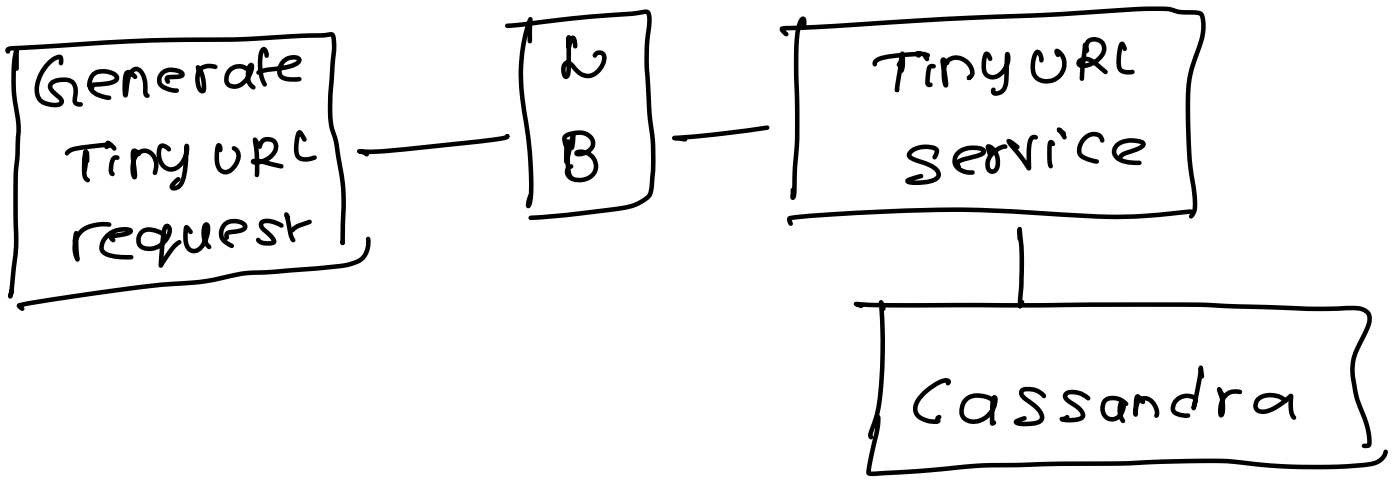
= n = 62^n

$$62^n > y$$

Applying $\log_{62} \Rightarrow$ $\boxed{n = \log_{62}^{y}}$

$\therefore$ Length $(n) = \log_{62}^{y}$

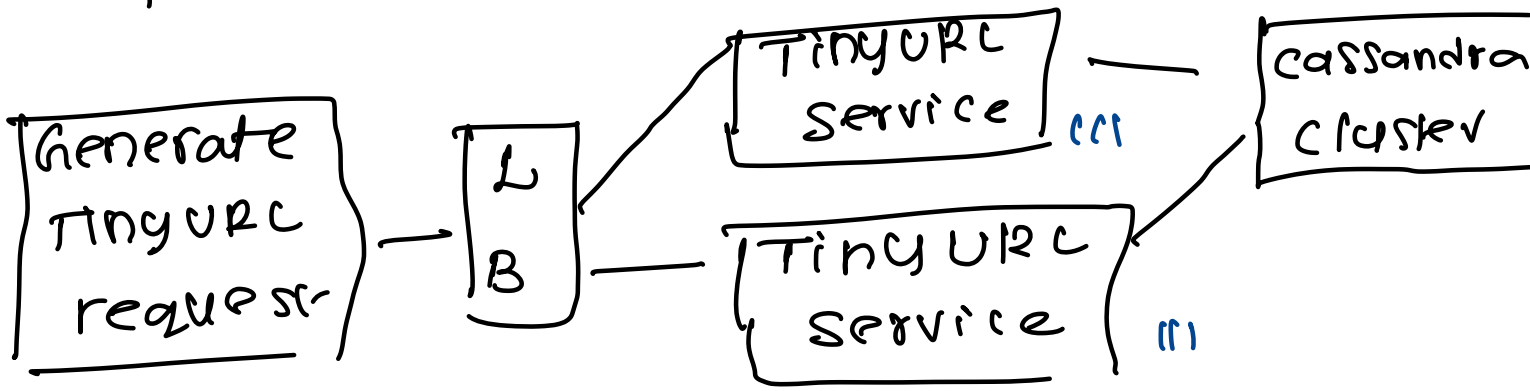
Basic Architecture:



(i) When a request to generate TinyURL comes to TinyURL service. It will generate and store in cassandra.

But the above design has issue i.e. Single point of failure

Therefore, scale up TinyURL service

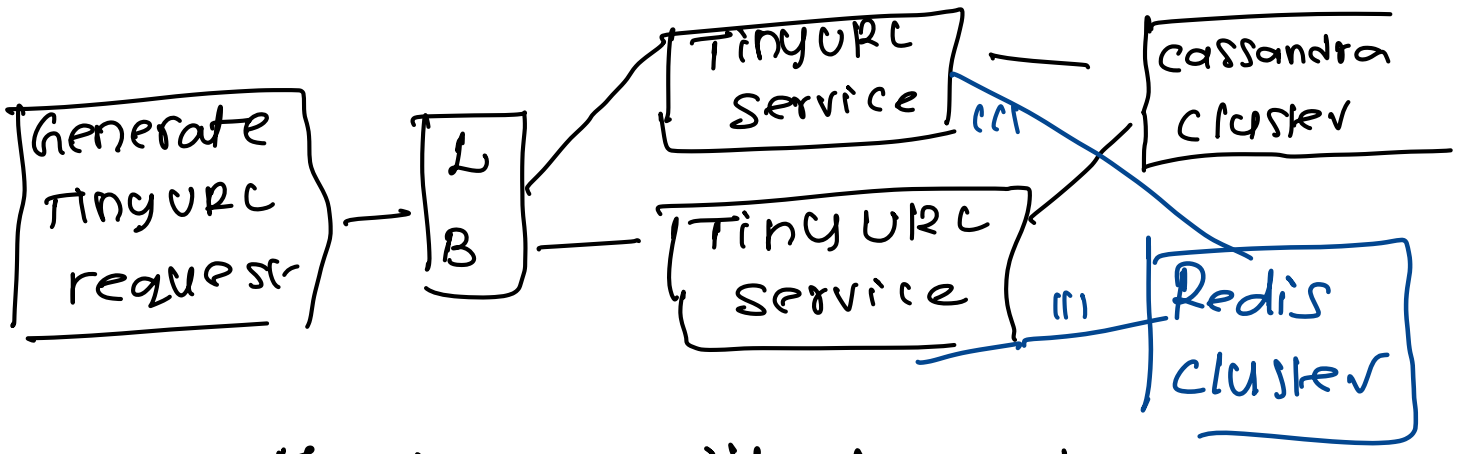


The issue with the above design is when TinyURL services start generating duplicate tiny URLs

Two long URLs cannot have single tinyurl

Eliminating duplication issue using Redis cluster:

By keeping an auto increment counter Redis cluster we can eliminate duplication

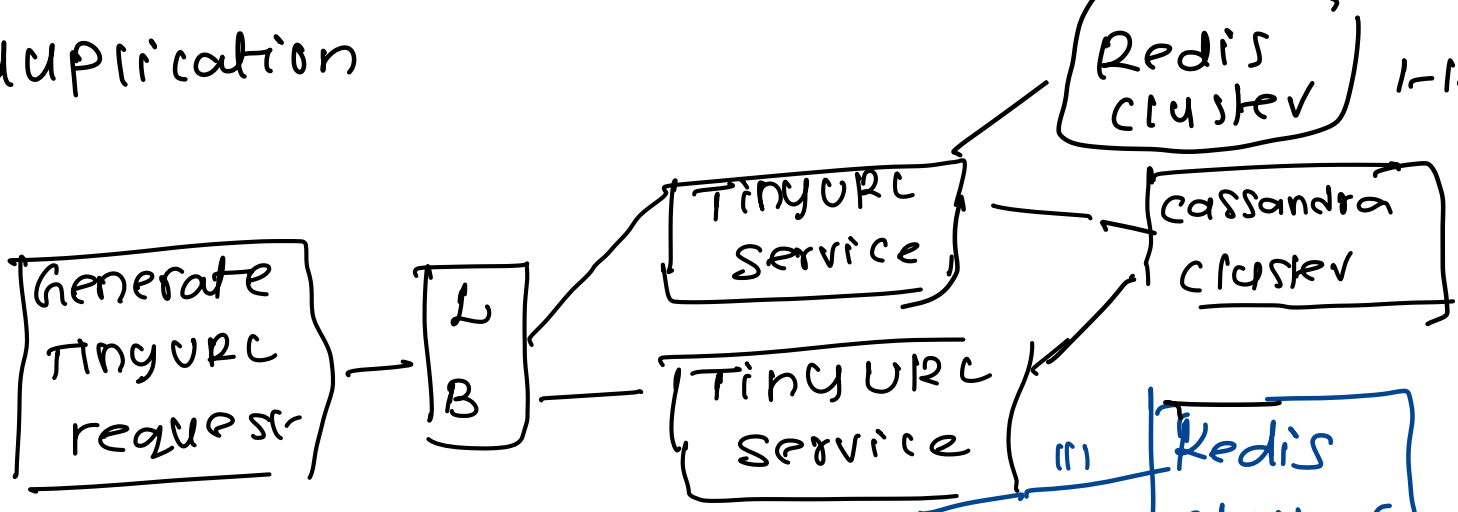


However, the issue with the above design is SPOF. If Redis cluster is dead whole system has to halt.

We can keep multiple redis clusters in multiple geographical locations and assign a set of TinyURL services to access nearest Redis cluster.

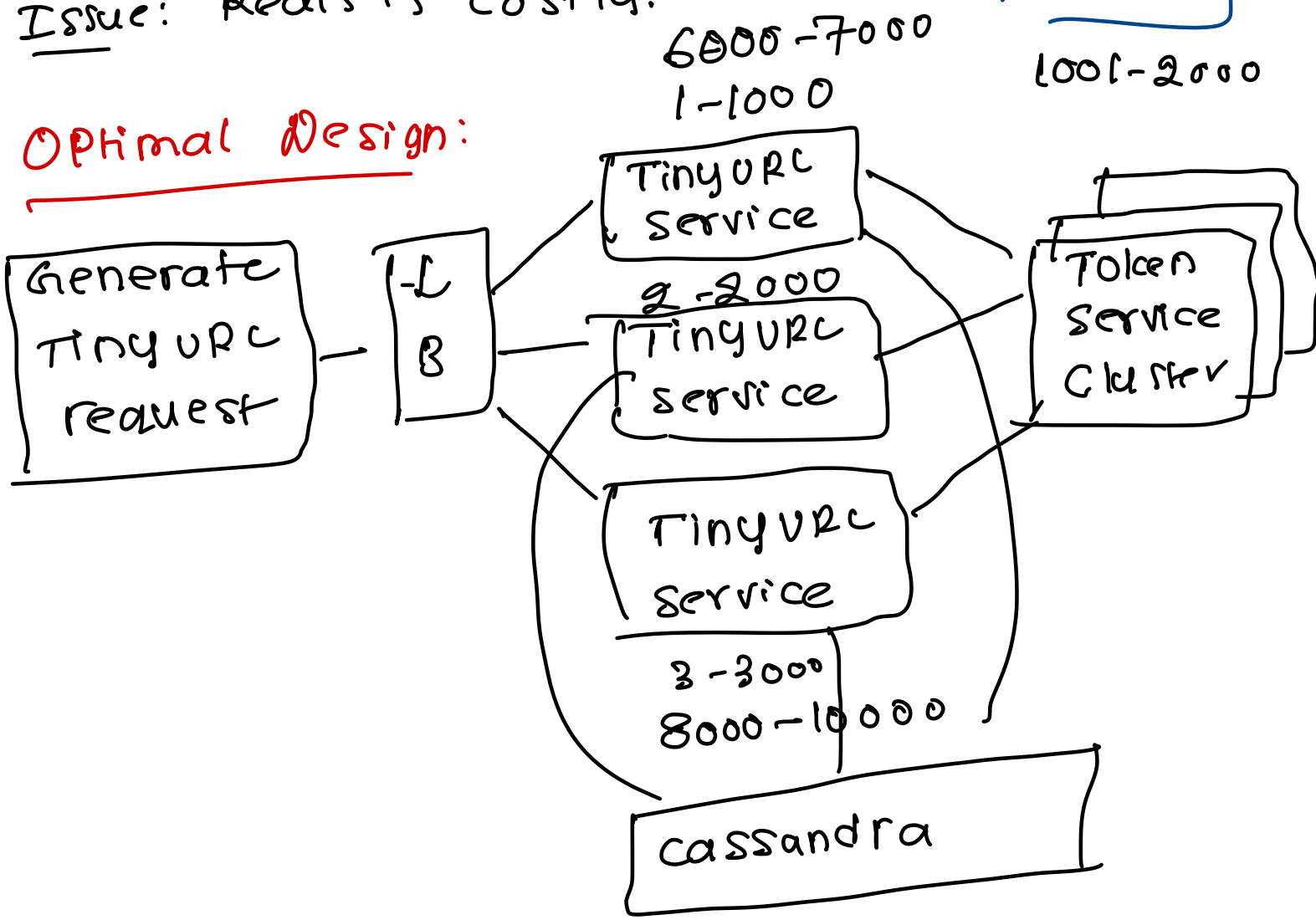
Each Redis cluster should have URL range say [1-1000], [1001-2000] to eliminate

duplication



Issue: Redis is costly.

Optimal Design:



By having explicity token service distributed cluster. which can assign a set of URLs range to every Tiny service we can eliminate duplicaton & cut cost without using Redis

DRY RUN:

(1) Generate TinyURL for long url request comes to TinyURL services. TinyURL services will have auto increment counters and range in which they can choose numbers from.

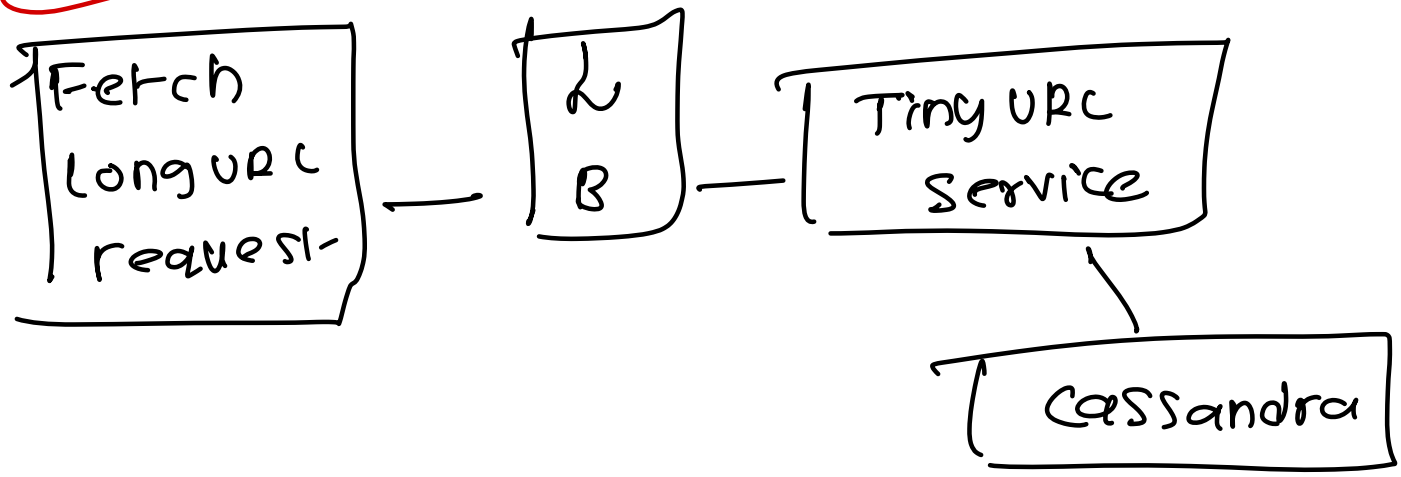
The range of URLs is set by Token service

(2) When threshold is reached -- Tiny URL's get new range from Token service

(3) When TinyURL service restarts the token range is lost and requests for new range from Token service

* Drawback

Fetch LongURL when ShortURL is provided:



When fetch long URL request comes to the TinyURL service ... It will fetch the long URL from Cassandra Database.

Why Cassandra:

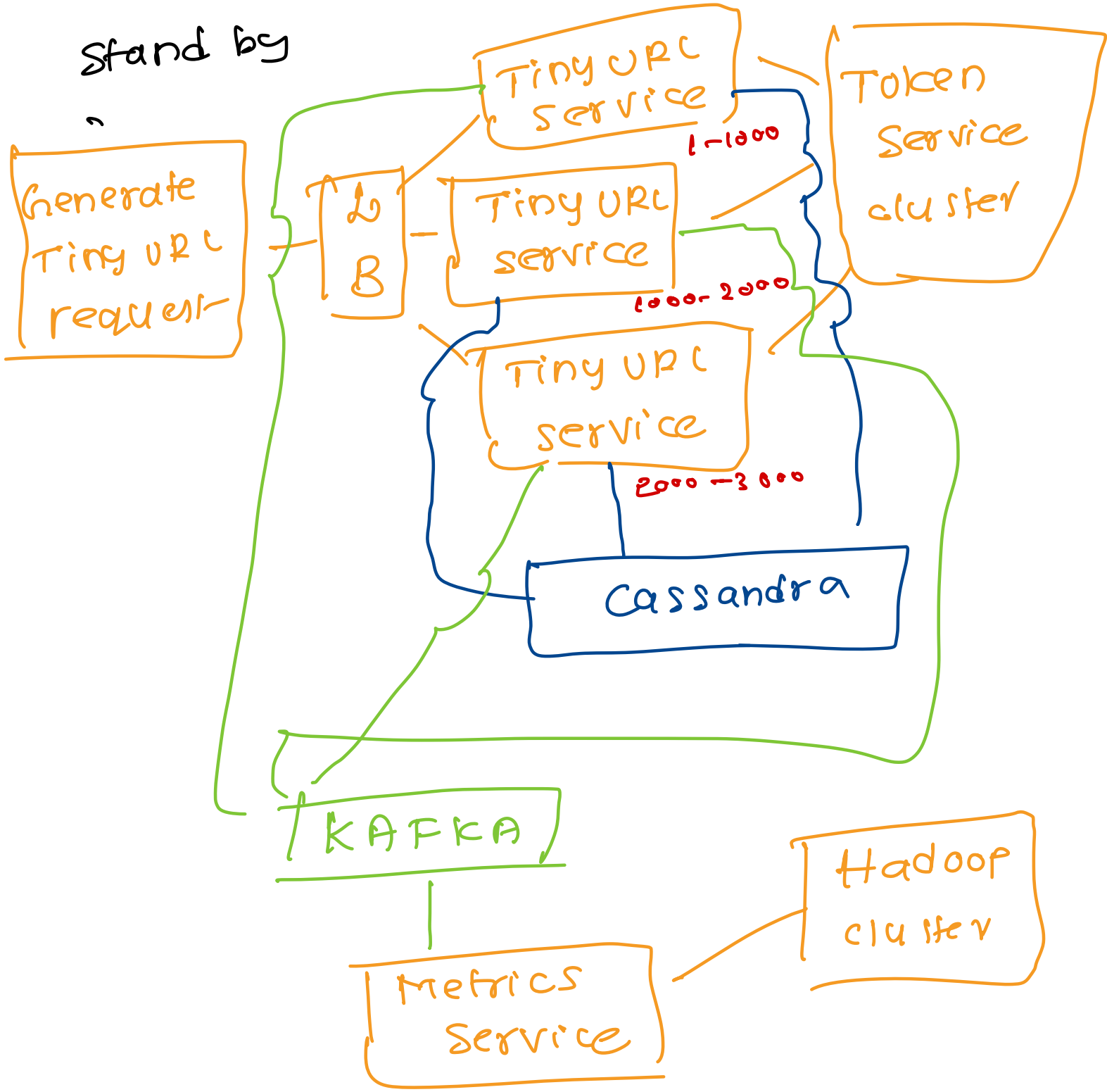
We can have mysql cluster that follows sharding as well but cassandra can handle 62^7 URLs writes and reads effectively. Therefore, cassandra is a better option.

Metrics:

(1) We don't keep track of any information like from which geography the following URL request comes from. From which device, which ip etc...

(2) If we push the requests into kafka topic. Then it will be really helpful in identifying metrics like which country makes more requests, which device and soon.

(3) with this information -- we can keep data centres near to that location as primary to eliminate latency and use other data centres as stand by



To eliminate latency, we can reduce I/O calls to Kafka. Instead of sending every request we can aggregate a threshold of requests and

push to kafka (probably a cluster of reques

Also, we can employee concurrent processing but not parallel processing for pushing messages in to kafka