Contents

[Horizontal v/s Vertical Scaling 2](#_Toc109199254)

[Distributed System 2](#_Toc109199255)

[Load Balancing 3](#_Toc109199256)

[Message Queue / Task Queue 3](#_Toc109199257)

[Monolith V/s Micro services 3](#_Toc109199258)

[Database Sharding 4](#_Toc109199259)

[Distributed caching 6](#_Toc109199260)

[Write –Through Cache 6](#_Toc109199261)

[Write –Back Cache 6](#_Toc109199262)

[Why we need to go for caching? 7](#_Toc109199263)

[Event Driven Services 8](#_Toc109199264)

[Request and Response Architecture 8](#_Toc109199265)

[Publisher Subscriber Model 8](#_Toc109199266)

[Content Delivery Network 10](#_Toc109199267)

[Case Study 11](#_Toc109199268)

[How Netflix onboards new content 11](#_Toc109199269)

[Netflix Caching Mechanism 12](#_Toc109199270)

[References 13](#_Toc109199271)

[Estimate Youtube daily video storage requirements 14](#_Toc109199272)

# References

|  |
| --- |
| <https://www.enjoyalgorithms.com/tags/system-design-concepts/> |
|  |

# Horizontal v/s Vertical Scaling

Scaling – The ability to increase or decrease IT resources as needed to meet changing demand is called Scaling

|  |  |
| --- | --- |
| Horizontal Scaling | Vertical Scaling |
| It is an ability to add more system to meet the demand | It is ability to add a big machine to meet the demand |
| Load balancers are required | NA |
| **Resilient – Ability to withstand or recover quickly** | Single point of failure |
| Network calls (RPC) | **Inter process communication** |
| Data inconsistent | **Consistent** |
| **Scale wells as users increases** | Hardware limit |

In real world application which scaling will use?

Typically it based on the requirement. But general idea is will go for hybrid scaling where we select some of the good features from vertical scaling like Inter Process Communication, Consistent and from horizontal scaling Resilient and scale well features.

# Distributed System

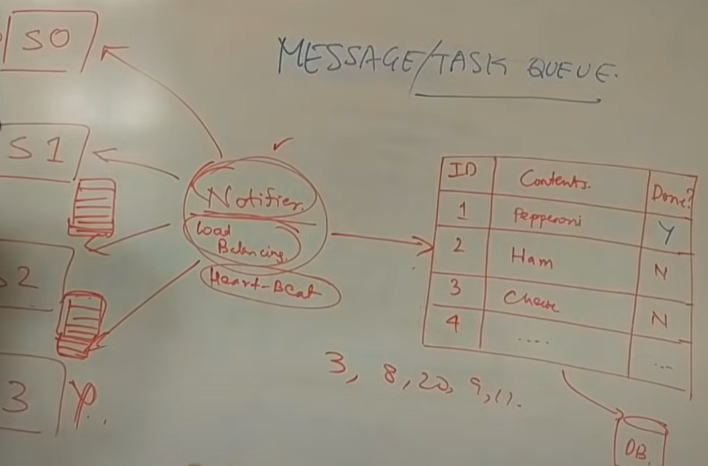
Taking measure to start with distributed system

* Optimize the processes and increase the throughput of the system with same resources – **Vertical Scaling**
* Preprocessing and configuring jobs
* Keep Backup and avoid single point of failure – **Master – Slave Architecture**
* Adding more resources as business growing - **Horizontal scaling**
* Micro services architecture – Providing particular responsibility for a task/system
* Distributed system - Partitioning
* Load Balancer
* Decoupling
* Logging
* Extensible

# Load Balancing

Consistent Hashing is the mechanism of balancing load equally on “N” Servers.

# Message Queue / Task Queue



References

<https://stackoverflow.com/questions/16715380/what-is-the-difference-between-asynchronous-and-synchronous-http-request>

<https://www.enterpriseintegrationpatterns.com/patterns/conversation/RequestResponse.html>

<http://blog.codepath.com/2013/01/06/asynchronous-processing-in-web-applications-part-2-developers-need-to-understand-message-queues/>

<http://highscalability.com/blog/2012/12/17/11-uses-for-the-humble-presents-queue-er-message-queue.html>

<https://www.cloudamqp.com/blog/what-is-message-queuing.html>

<https://www.rabbitmq.com/getstarted.html>

# Monolith V/s Micro services

|  |  |
| --- | --- |
| Monolith | Micro services |
| It is an application with huge machines(one or more) which allows the client to connect and processes there requests  **Note**: Monolith is a single huge machine running entire system, assumption is wrong. | It is a single business unit, where functions related to specific business put together as a service and which are responsible for communicating its own databases. |
| Client will directly talking to the services | Client is talking with the gateway and geteway talks with services internally. |
| Pros:   * Works well with small team * Less moving parts or less complex * Code reusable / less duplicated code, in turn when we are implementing we features * Performance is faster and we are not making any network call within the services (no RPC call)   Cons:   * Required system and business knowledge for new members to start working with as everything is a single business unit * Deployment is complicated, as everything needs to deploy even for a single change. * Much dependency on single server, in case of anything goes wrong, entire application will go downs.   e.g) Stack Overflow | Pros:   * Easily scalable * Easy to design * Easier for the new members to understand and start with * Supports parallel development as a team can work on analysis, reports, chats etc * Easy to scale particular services, support if chart requires more services, we can scale particular service.   Cons   * Required skills to design a better system architecture   e.g) Amazon, flipkart, google etc |

# Database Sharding

References

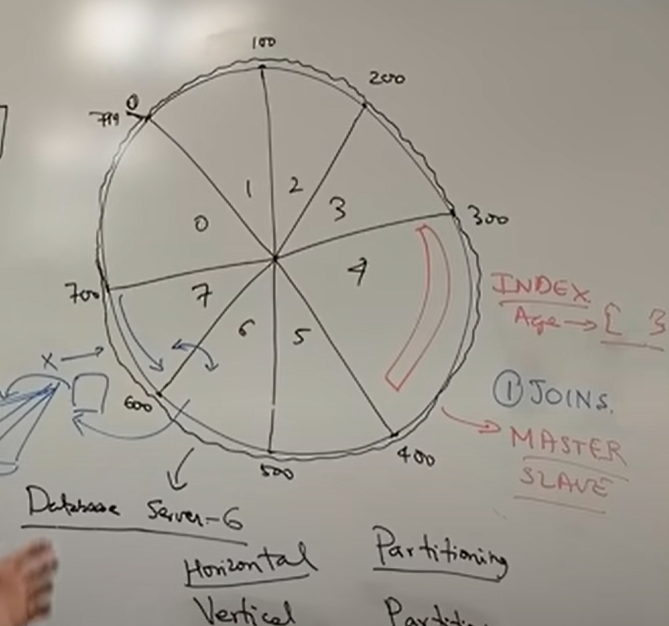
<https://en.wikipedia.org/wiki/Consistent_hashing>

<https://medium.com/@jeeyoungk/how-sharding-works-b4dec46b3f6>

<https://medium.com/@jeeyoungk/why-i-love-databases-1d4cc433685f>

<https://www.quora.com/What-is-the-difference-between-consistent-hashing-and-sharding-in-a-distributed-system>

<https://en.wikipedia.org/wiki/Partition_(database>



Horizontal partitioning v/s vertical partitioning

Cons

* Joins across the shards are expensive: As each shards maintains in its own network, Quering across the shards are going to be expensive
* Fixed number of shards
* We can go for Master Slave architecture to avoid shard failures. (As we know, Master Slave architecture is the process of taking the immediate backup of database as write operation are always performs on Master database whereas Read operation are perform on Slave)

# Distributed caching

Policy: The way of loading and evocating data is call policy or cache policy

LRU - Least recently used policy

LFU – Least frequently used policy

Sliding windows base policy

Types of Caching

1. Application cache
2. Server side cache

Server side cache

* Placing cache close to server will help to improve read and write operations

Cons

* If the server fails, the cache associated with the server also get fails
* If cache on server S1 and S2 are not same or not in sync. It may lead to bad user experience

Global Cache

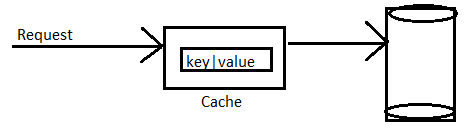
* Placing a global cache were request from any server will first hit to the cache, if data is not found then system will run query against the server

Pros

* More accurate, even in case of any server fails. As we are maintaining data in a separate global cache
* We can scale the Redis cache separately

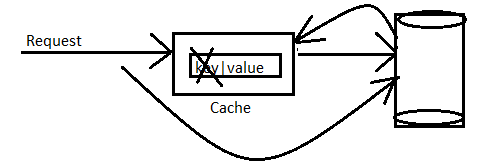
## Write –Through Cache

* It is the process of writing the data (or updating) into the cache before updating the data saving into database.



## Write –Back Cache

* It is the process, when the request is send to cache, it will delete the data associated with that key and send the request to database. In return from database, data will be updated in cache and send the response.

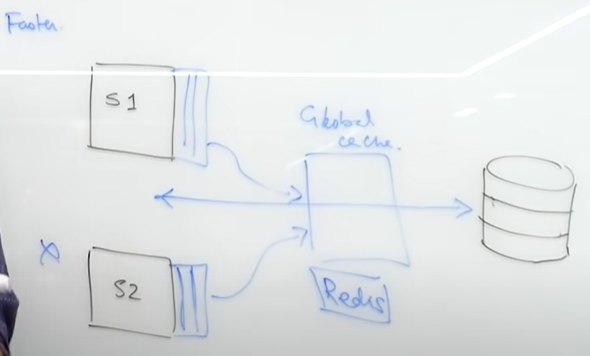


Pros and Cons

* Write-through caching, whenever we are writing the data into cache before saving to database may result in consistent in case of having multiple servers (say S1 and S2). In typical situation cache on one server will remains in consistent
* Write-back cache, as we are always writing the data into database first and then updating the cache, it may result expensive and take more network calls.

Use cases: In real world, if we are dealing with any financial data or user credentials go with write back cache.

If we are dealing with general updates like updating notes, comments etc. for work write-through cache and persists all the entries on database periodically (say for each 30 seconds)



## Why we need to go for caching?

* To reduce network calls
* Avoid re-computations
* Reduce load on Database

References

<https://github.com/google/guava/wiki/CachesExplained>

<http://www.mathcs.emory.edu/~cheung/Courses/355/Syllabus/9-virtual-mem/LRU-replace.html>

<https://en.wikipedia.org/wiki/Cache_replacement_policies>

<https://github.com/ben-manes/caffeine>

<http://highscalability.com/blog/2016/1/25/design-of-a-modern-cache.html>

# Event Driven Services

## Request and Response Architecture

* In the below simple micro service architecture, we can see that. When a request is made by the device, the request is first goes to server S1.
* Server S1 will delegate the request to S0 and S2. From S2 request will get delegate to S3 and S4.
* Here we can see that, here S1 has to wait till S0 completes the processing and return the response.
* Similarly S2 has to wait for S3 and S4 to complete the process and return the response

Using Request and Response architecture will not give the better result. The smart thing we can do it to make the asynchronous calls.

Even asynchronous calls also have the similar dependencies.

Cons

* Strongly coupling
* Failure Latency - Assume if S4 fails. It will take a long time to return the response as S1 and S2 are pass.
* Inconsistent Data

To overcome the above challenges of Event driven Services we can go for Publisher Subscriber model

## Publisher Subscriber Model

Introducing a message broker like Kafka or RabbitMQ to deal with request and response will going to solve above problem.

Message brokers are persistent which takes the responsibilities for handling the event at some point of time.

Pros

* Decoupling the system responsibilities
* Easy to understand and single point of failure
* Some transaction capabilities
* Easily scalable. Like introducing a new server S6 is just need to be subscribe for S1 to get into system

Cons

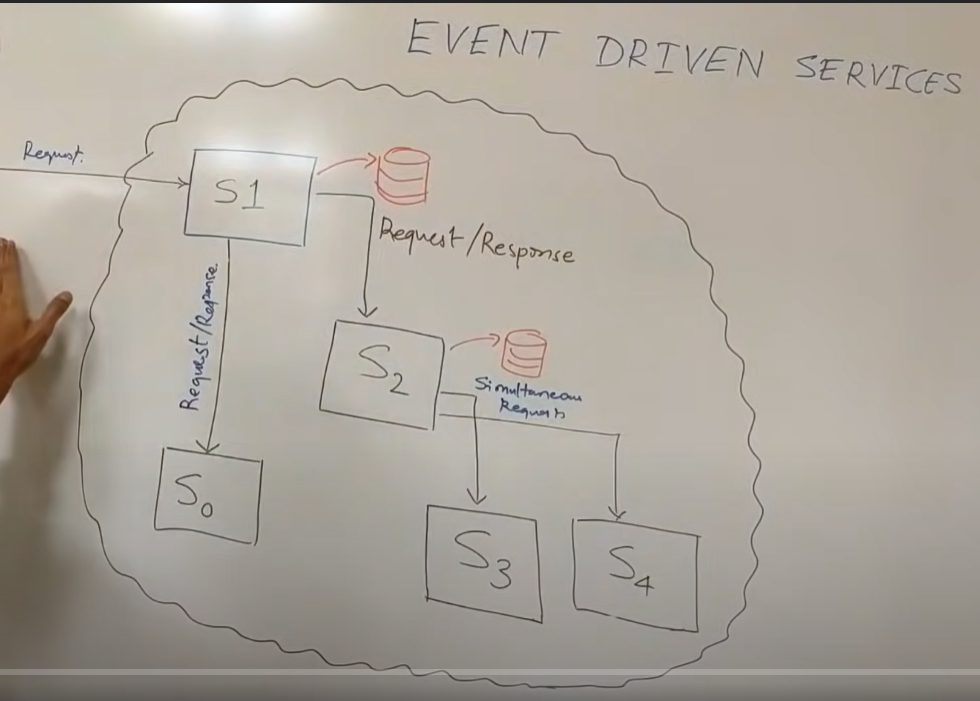
* Not recommended for the system which requires more consistency
  + Like financial application, banking application where Invoice and Fund transfers cannot be maintained as two different services.
    - As if invoice may generate first by deduction of service tax but fund transfer service fails to process.
    - Handling multiple deduction may result in consistent for independent service like S2

Publisher and Subscriber model works well for gaming services.

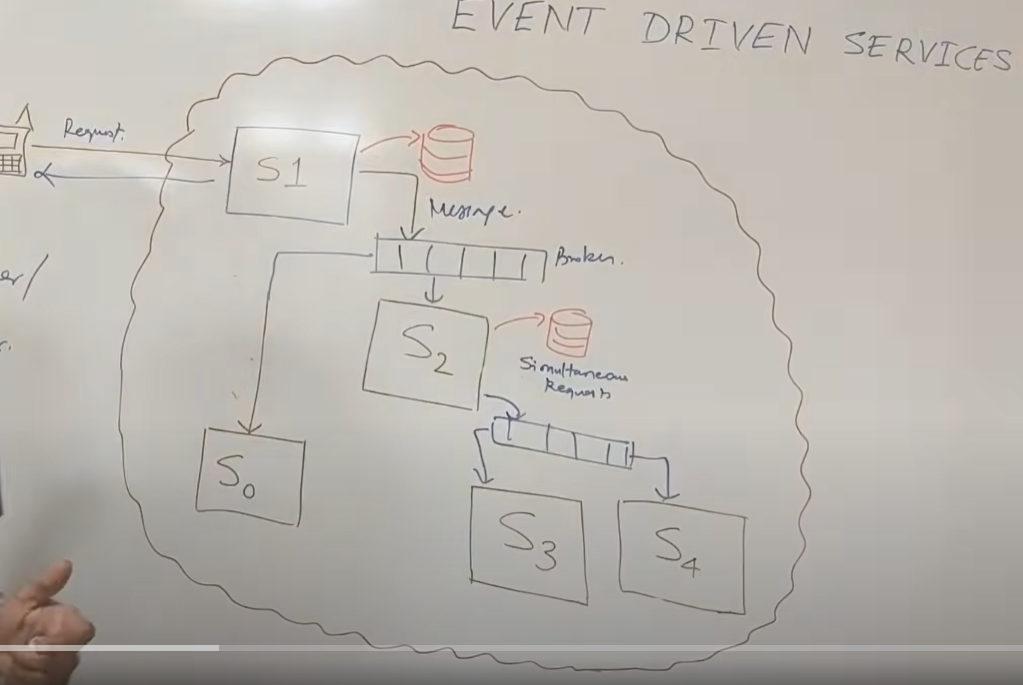
Not recommended for financial application

Twitter follows the Publisher subscriber model as a twit can consume by any subscribers.

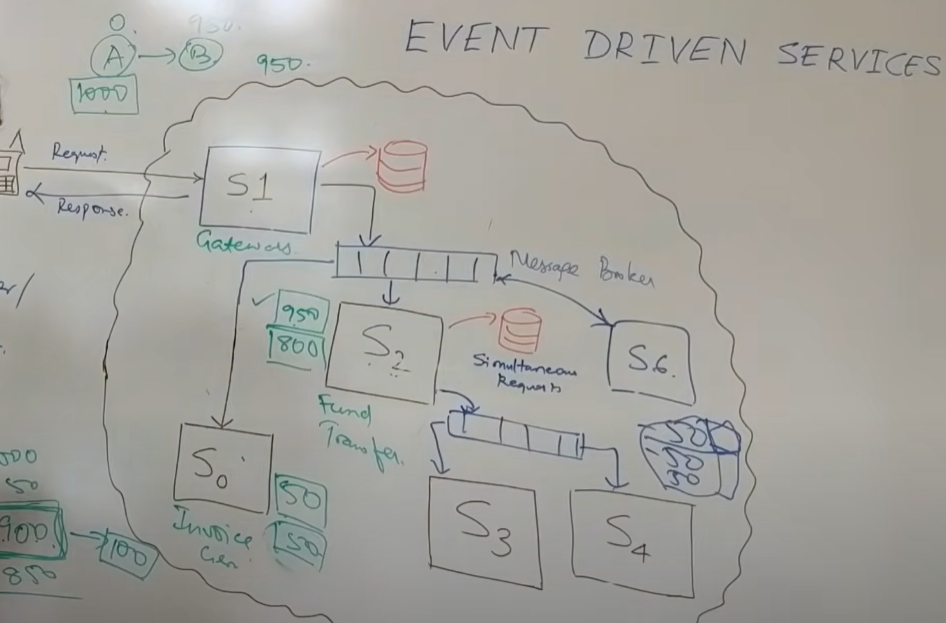
**Request & Response Architecture**



Event driven architecture



Cons – Not application for financial service



References

<https://docs.oracle.com/cd/B10501_01/appdev.920/a96590/adg15pub.htm>

<https://aws.amazon.com/pub-sub-messaging/>

<https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern>

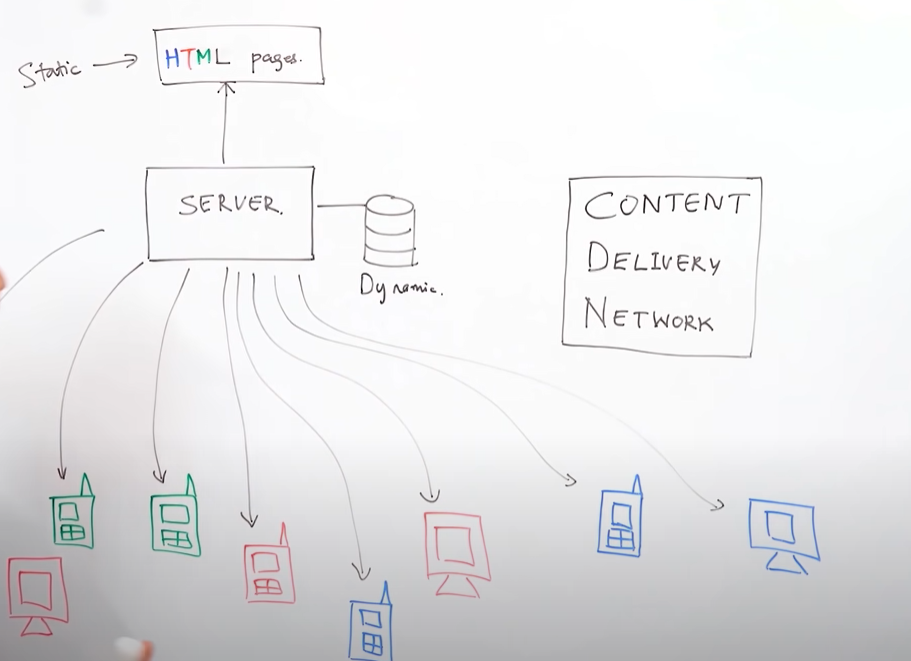
# Content Delivery Network

Whenever a user make request from the devices from different regions, to the server which internally coordinates with application and render the specific page

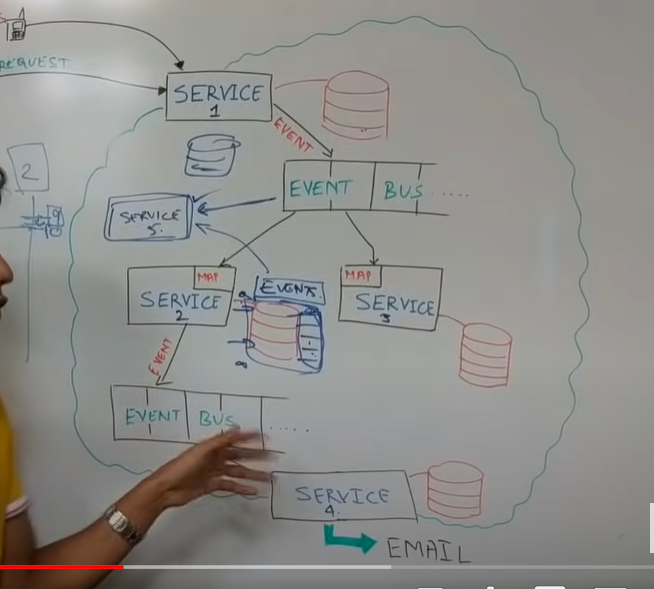
Cons:

* Data Path is long
* File type depends on user device

Content Delivery Networks are a bunch of servers spread across the globe to serve information. These networks are available on rent to deliver static content quickly to nearby users. Some examples of CDNs are Amazon CloudFront and the Akamai CDN. They are (relatively) cheap to rent and have high availability. They also provide pluggable algorithms to invalidate and fetch data.



# Event Driven System



Pros:

* Availability
* Easy Roll back
* Replacements
* Transactions
* Stores Intent

Cons:

* Consistency
* Not Applicable to Gateways -- replacing the services which stores external services response like Email service in our case is not easy. (as it stores other service responses and timestamp)
* Lesser Control
* Flow of program is not easily understandable
* Not easy to move to other architecture like moving to request response architecture

e.g) GIT, gaming application uses event driven architecture

References

<https://www.youtube.com/watch?v=STKCRSUsyP0>

<https://martinfowler.com/articles/201701-event-driven.html>

<https://en.wikipedia.org/wiki/Event-driven_architecture>

<https://microservices.io/patterns/data/event-driven-architecture.html>

# Case Study

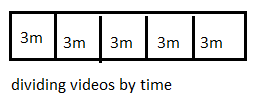
## How Netflix onboards new content

**Different formats**: Need to maintain different quality of videos like High, Medium, low etc to target different types of user experiences

**Different Resolution**: Need to maintain different resolutions like 1020 px, 720 px to target different devices like TVs, Laptop, Mobiles etc

Compressing the video quality is the process of reducing the size of the file.

* Suppose we build a process where we provide 6 GB of file to the system to compress it around 1 GB. These process of compress is typically work out, but depending on only one system to process the file may result in facing lot of challenges like
  + Waiting process is more
  + If system crash, will lose the file which are in process
  + Not a good idea to compress entire file once as it may result in last minute failure
* Need to divide a big file into small crunches
  + Dividing a file into small crunches will be a better approach, but the parameter we choose to divide will need to choose as per requirement.
  + **Dividing by Time**, like we maintain 3 mins of crunch and will allow the tasks to take care of independently.
    - Cons:
      * As we are dividing the video by time, assuming a scene of car racing, user will interrupted in between as it have to wait for another crunch after 3 mins. Which is leads a very bad user experience.



* **Dividing by Scene,** when we divide a crunch by scene where each crunch might be part of 30 secs or more, will help us for a good streaming quality where each task is responsible for loading each crunch.

**Prediction Algorithm**

* This algorithm is clever which study user habits and process as required. Suppose if we are just clicking on a streaming (processing) bar at randomly and viewing a particular scene, then this algorithm is clever enough to load the particular content instead of loading the other streams in advance.

**ISP - Internet Service Providers**

* It is an organization that provides services for accessing, using or participating in the Internet.
* It is a place where we have address registration for each site. Like if we are requesting for the facebook, gmail etc. it will typically maintains the address to access those machines

### Netflix Caching Mechanism

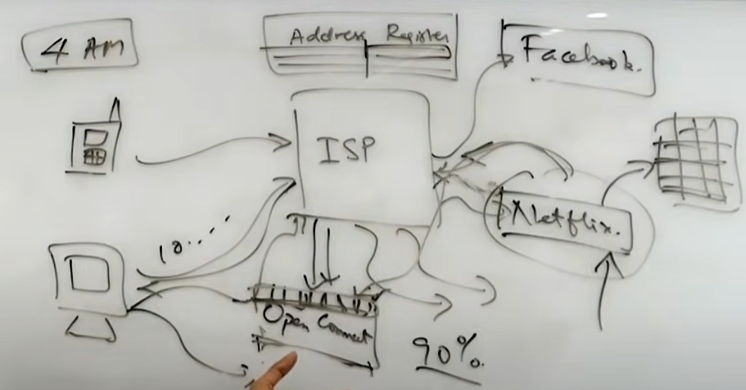
In a typical scenario, if the system are hosted in the U.S, to carry each and every user request to the USA will be typically difficult, especially the video streaming services.

To gain the better user experience and avoid the round trip for each and every request over the ISPs, internally Netflix will cache the content.

Whenever there is a request from the users, Netflix algorithm will first request the cache data and process the response, if the requested data is not available. System will request the hosted server.

Jobs

As part of daily processing, there will be an algorithm which responsible for caching the trending or latest video into specific region servers.



### References

<https://netflixtechblog.com/high-quality-video-encoding-at-scale-d159db052746>

<https://netflixtechblog.com/optimized-shot-based-encodes-now-streaming-4b9464204830>

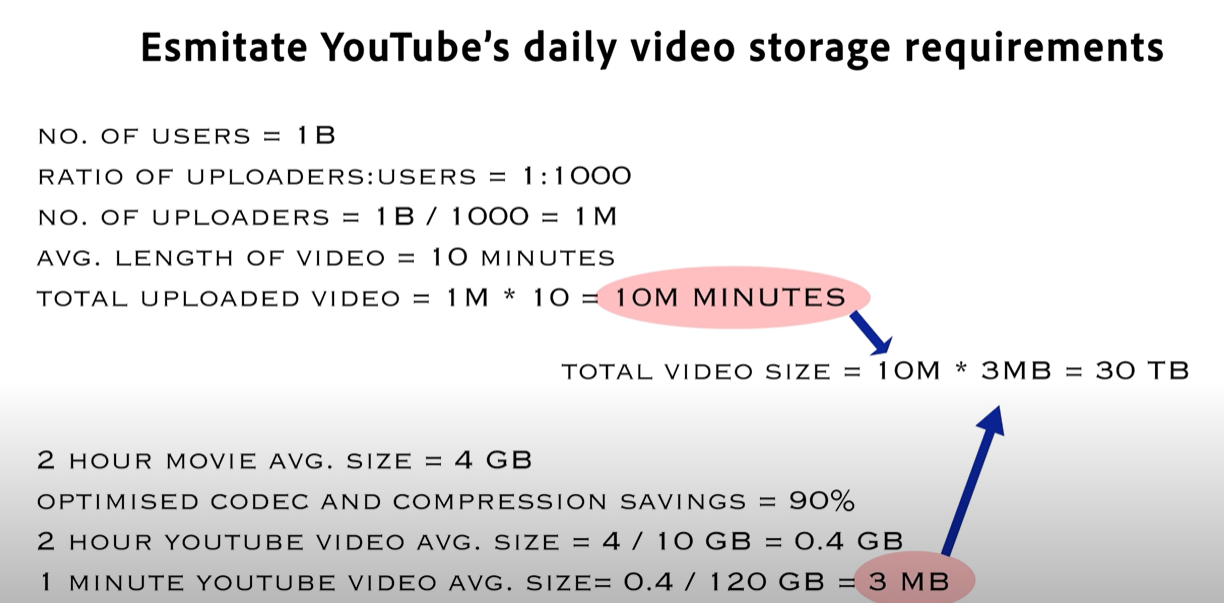
<https://netflixtechblog.com/tagged/video-encoding>

<https://netflixtechblog.com/tagged/stream-processing>

<https://netflixtechblog.com/keystone-real-time-stream-processing-platform-a3ee651812a>

<https://about.netflix.com/en/news/how-netflix-works-with-isps-around-the-globe-to-deliver-a-great-viewing-experience>

## Estimate Youtube daily video storage requirements



References

<http://highscalability.com/youtube-architecture>

<https://www.youtube.com/watch?v=w5WVu624fY8>

<https://www.youtube.com/watch?v=modXC5IWTJI>

<https://www.youtube.com/watch?v=G-lGCC4KKok>

<https://www.youtube.com/watch?v=-3qetLv2Yp0>