# Requirements

**Top K Heavy Hitters**

**APIs**

Functional : 1) Given user views count for a video, ﬁnd top k most frequent visitors

2) Top k visitors for given time duration which can be max 1 year Non Functional: 1) Highly available, Eventually consistent

2) Scalable, should be able to work for geographically distributed data.

# Assumptions and Capacity estimation



User/Visitor

Load Balancer

Web Servers

VisitorCounter Service

1. Assume 1M views per day for trending videos. Assume 100M registered users.
2. ~1500 views/min. Assume 5-10 views/user => 0.1M unique users.
3. UserID<String> getTopKVisitors( videoID<String>, startTime, endTime, k <Int>)
4. incrementCount( videoID, userID, currentTime )

**HLD**



1. Maximum time duration can be 6 months -> Assuming eventual slowdown of visit distribution -> need to store data for ~ 0.1M x (30 x
2. days x 100B which is ~ 120GB/video

Observations - 1) The service invocation needs to be async since web server's main task is to load the content. The service is basically

# In-memory based approach (users<1k)

**Cloud CDN**



used for analytics

1. No of writes is very frequent compared to read results from the service.
2. Since the read API is not customer facing, some processing can be done at runtime before getting ﬁnal results.
3. Due to large data/video, need to use partitioning and parallelization for computation.

# Approach using DB(users<100k)

* 1. Instead of directly sending to counter processor, add the visitor details to a message queue.
  2. The processor picks the user details, from the queue and update minHeap and counter.
  3. For every elapsed time interval add the top k visitors to DB.

Use data structure which is combination of minHeap and Counter Map. Algo -

1. Initialize a minHeap and a counter map
2. For each user id do
3. If count (userid ) > top(minHeap) then
4. remove top of minheap and add new userid to heap
5. After every ﬁxed time interval, add top k users to a result queue, also decrement view counts of users whose time was outside current window.

Problems with this approach for large no of users -

1. Cannot be used for large #userids which can't ﬁt in memory
2. Historical details difﬁcult to obtain.
3. Cannot distribute load to multiple servers. Best case possible is to put heap on one server and counter map on another.
4. NoSQL can be used as DB since it ﬁts into k-v structure, preferable HBase since it supports fast multiple writes.

HBase



Kafka MQ

API Gateway for

the service

Distributor/LB

Web/App server



MapRduce cluster

# Control Flow

Results Computor

* 1. The web server creates a batch of view counts /video and when load is low, sends it to API gateway.
  2. The API gateway creates message, assigns topic to the message and adds to Kafka. The Topics are based on range of UserIDs.
  3. The distributor picks the message from queue, based on topics, sends those messages to matching destination node. The nodes are partitioned by range of user ids.
  4. The Mapreduce cluster computes the aggregated results i.e Pair<time ,Pair<userid, viewcount,> and stores the results on HBase. The HBase stores the results where key is timestamp and value is list of users during that time interval ( mostly 5 min interval). The results are stored to HBase for each ﬁxed time interval.
  5. HBase is used since it supports fast writes, schema ﬂexibility and nicely ﬁts with MapReduce.
  6. The result computor reads the data from HBase, based on input time range, ﬁnds viewcount of top users by using cumulative's difference approach, ﬁnds top k users from those users and returns the results.

Results

Request (getTopK)

Client