https://www.youtube.com/watch?v=kx-XDoPjoHw&ab_channel=SystemDesignInterview

Simple approach
per node, get top k values
then perform k-way merge of k sorted arrays

Non-functional requirements

- Scalability
- Perforamnce
- Availablity
- Accuracy

who pays the cost?

- CPU vs memory vs IO
- if you use a complressed binary data format eg avro
  - good for IO
  - but CPU will have to pay the cost of deserialization

Count-min sketch

"the absence of replication can greatly simplify a system"

- sometimes it does not matter if we lose data
- so then just use memory on a small number (maybe even just one) of hosts

Everytime you partition you need to think about

- hot partitions - load balancing
- replication - for availability
- cluster rebalancing - what to do when you add/remove a node

Distributed messaging system

- Kafka, Kinesis

A good general design practice is to keep aggregating and/or batching data at each step of the pipeline, so the subsequent stage in the pipeline has to deal with less scale

- eg: when user hits API gateway -- API gateway can keep a count aggregation in memory in a hash table until it gets too large -- then it can send the result of the hash table count to a distributed messaging system for futher processing.
- so the API gatway gets one request per view, but the subsequent pipeline only gets batched/aggregated requests: one per many n views

Lambda architecture

- combine a stream processing and a batch processing system
- makes the system very complex