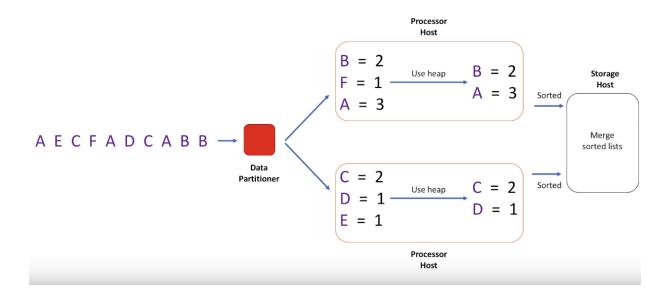
Top K Frequent Hitters

Return a list of most viewed videos for the last several minutes Stream processing problem

topK(k, startTime, endTime)

Single host:
Load into memory
Hash table —> sort or heap
Leetcode TopK algorithm

min heap: size K



Processor hosts only pass a list of size k to the storage hosts We don't pass the whole hash table

Q: streaming data is unbounded, infinite

Storage host stores a list of heavy hitters for every minute

Q: what if we want to know heavy hitters for 1-hour or 1-day period?

Need the data for 1 day

Store all the data in disk and use batch processing framework to calculate

MapReduce

Data Partitioning: data replication

Each partition are stored in different nodes

Rebalancing: when a new node is added to the cluster or removed

Simpler solution:

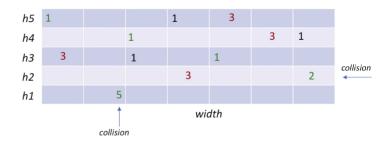
Using fixed size memory but the result is not 100% accurate

Count-min Sketch

Replace the hash table which can grow big with a count-min-sketch that always has predefined size

Count-min sketch





How is data retrieved?
For example, what is the returned value of A?

Minimum of all the counters for the element.

How do we choose width and height (number of hash functions)?

N/A

 \emph{e} - accuracy we want to have \emph{d} - certainty with which we reach the accuracy

width = Math.ceil (2.0 / e) height = Math.ceil(-Math.log(1 - d) / Math.log(2)) Count-min sketch returns frequency count estimate for a single item. Where is a top-k list in this picture?

Keep a heap of top-k elements.

High Level Design

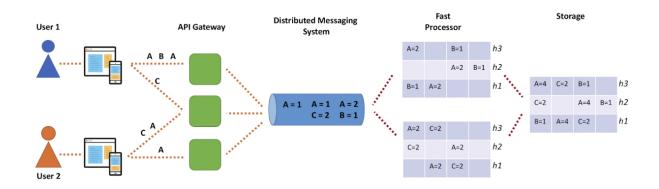
Fast Path: count min sketch Slow Path: HDFS, MapReduce

2 MapReduce Jobs: Frequency Count MapReduce Job, Top K MapReduce Job

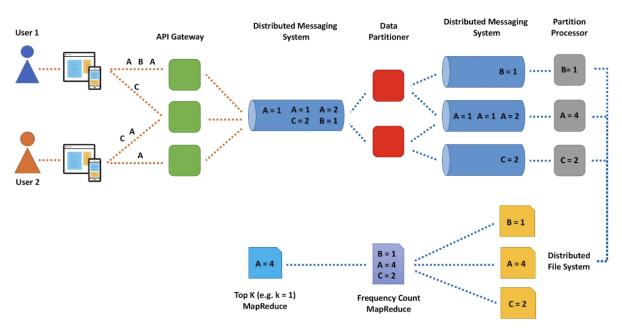
Something faster than MapReduce: partition again

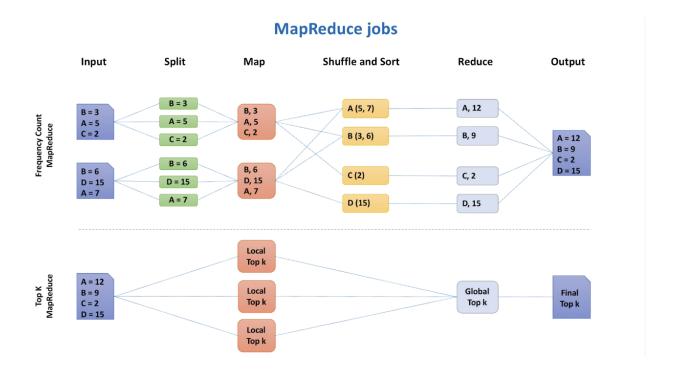
API gateway hosts: log the requests; hash table: count of views (# times viewed / video in last several seconds) aggregated on each host Information about the views are sent to the distributed messaging system

Data flow, fast path



Data flow, slow path





Data Retriever:

Data retrieval

