Ad click event aggregation

Digital advertising = real time bidding (RTB) = digital advertising inventory is bought and sold

speed -> occur in less than a second
accuracy is also very important.
important key metric: click-through-rate (CTR) and conversion rate (CVR) depend on aggregated ad click data.

Problem and design scope

1. Format of input data

log file append in different server, latest click append on end, with input (ad_id, click_timestamp, user_id, country)

2. Data volume

1 mn ad click per day, grows 30% over year

3. Imp query to support

return number of click for an ad in last M minutes
top 100 most clicked ad in past 1 min, both param should be config.
support data filtering by ip, user_id, country

4. Any edge case?

there might be event that arrive later than expected
might have duplicate event
different system can be down at anytime, consider system recovery

5. Latency requirement

a few mins - used for billing and reporting

functional requirements

- return number of click for an ad in last M minutes
- top 100 most clicked ad in past 1 min, both param should be config.
- support data filtering by ip, user_id, country
- dataset volume at fb, google scale

non function requirement

- aggregation correctness
- proper handle delay and duplicate event
- end to end latency - less than few minutes
- robustness - system should be resilient to partial failures

Back of envelope estimation

1 bn DAU

QPS = 10^9 event / 10^5 second in a day = 10000

peak ops = 5 times = 50000

storage = 1 click = 0.1 kb storage - daily storage = 0.1 kb* 1 bn = 100 gb, monthly = 3tb

Propose HLD and get buy in

Query API design

- return number of click for an ad in last M minutes
- top 100 most clicked ad in past 1 min, both param should be config.
- support data filtering by ip, user_id, country

we need 2 API for this

API 1: aggregate number of clicks of ad_id in last M minutes

eg: GET /v1/ads/{:ad_id}/aggregated_count?from & to & filter

eg filter = 001 -> filter out non-US clicks

o/p = ad_id, count

API2: return top N most clicked ad_ids in last M minutes

eg GET /v1/ads/popular_ads ? count(top N most click ads) 7 window (aggregation window size in minutes) & filter (identifier for different filer strategies).

response = ad_ids

data model

we have raw data, then aggregated for every minutes

for ad_filter - add filter_id

Raw data	Aggregated data
Pros: full dataset, support data filter and recalculation	Smaller data set, fast query
Cons: huge data storage, slow query	Data loss, since derived data
Keep this for debugging, also to recalculate aggregate data in case of corruption	Keep this, as querying raw data will be slow.
Serve as backup data, move old data to cold storage	Serve as active data. Tuned for query perfomance.

<u>Choose right database</u>

Raw data - for user response prediction, behavior target, relevance feedback
since QPS = 10000, and peak QPS = 50000 -> write heavy

relational DB - write challenging
noSQL dbxassandra or InfluxDB - more optimized and for time range query, or AWS S2 for data format like ORC, parquet, AVRO

here we will use Cassandra - auto refresh dashboard - since aggregation every minute

High level design

input = raw data (unbound data stream), output = aggregated result

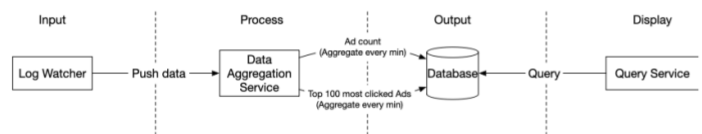

Figure 2 Aggregation workflow

above design in synchronous - not good if producer / consumer capacity is not equal

solution = kafka to decouple producer and consumer

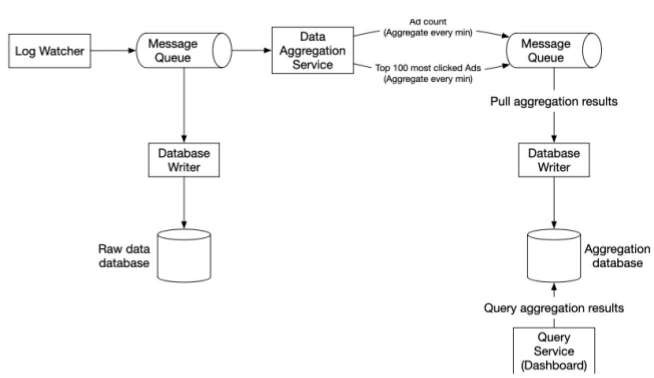

Figure 3 High-level design

Aggregation service

MapReduce framework - to aggregate ad clicks - use directed cyclic graph (DAG) - break system into small computing unit (map / aggregate / reduce nodes)

NOTE: an alternative to map is kafkaesque partition or tags and let node subscribe to kafkaesque directly, but here input data need to be cleaned or normalized and with map we don't need it.

NOTE: reduce node reduce aggregated result from all age node to final result, eg 3 aggregation node using heap dsa contain top 3 click ad within node -> reduce will reduce total number of most click ad to 3

DAG represent well known mapreduce algorithm

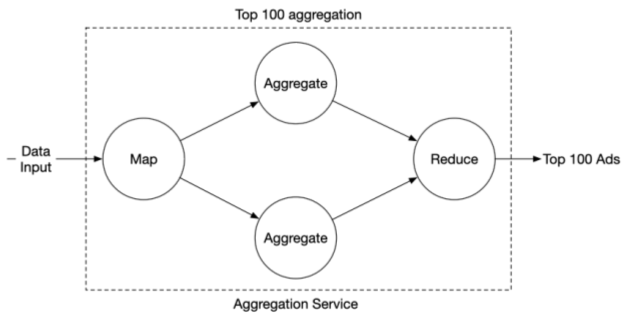

Figure 5 Aggregation service

Data filtering

use star schema to filter data with predefined criteria and aggregated based on them.

Design Deep Dive

Streaming vs batching

both are used -> call lambda architecture - stream to process and generate aggregated result in near real time, and batch for historical data backup.

kappa architecture - combine batch and streaming in one processing path

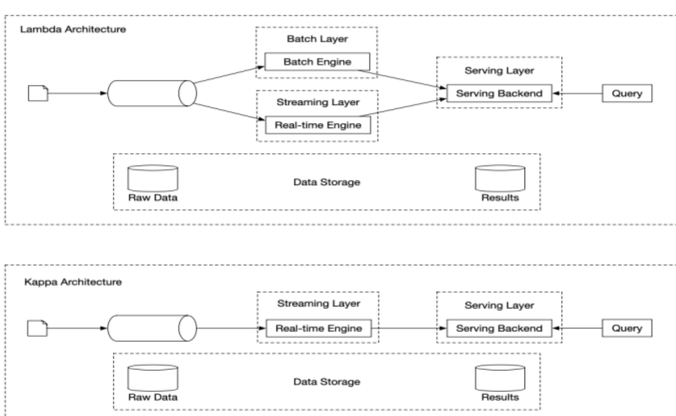

Figure 10 Lambda and Kappa architectures

Our high-level use kappa architecture - since sometimes need to recalculate age data from raw data starting option where error introduced

timestamp - to perform aggregation. Can generate at (event time when click happen, or processing time when egg server process click event)

with event time- age result are more accurate, but client might have wrong time or generated by malicious user

with processing time - server timestamp is more reliable but not accurate if event reach at later time

since data accuracy is important - use event time

- to handle delay event - use water mark to extend the time aggregation window, but be careful with time for watermark - large watermark - more latency; short watermark - less accurate data.

aggregation window

we have 4 window - tumbling (fixed), hopping, sliding, session window

tumbling window - fix - non overlapping chunk - good fit for aggregate every minute

sliding window - can be overlapping eg: sliding 3 min window run every minute to get top click ads.

delivery guarantee

exact once delivery semantics - since with at least once we might get duplicated which impact in million dollar

data deduplication

1 approach - use external file storage eg HDFS or S3 to record offset, and aggregator will process event only if last offset stored in storage sis not present

issues - we are storing before aggregation sent to result downstream, in case of aggregator outage - event will never be processed

solution - save in pdfs once we get acknowledgement back from downstream

Scale system

message queue - producer are easily scaled, consumer rebalancing happen in consumer group where we can add/remove nodes, since this might be slow, try to do this during off peak hours.

brokers -

hash key - ad_id hash key to store events with same id in 1 partition

partition number - allocate enough partition in advance to avoid dynamic increase in partition

topic physical sharing - can split data by geography or business type

aggregation service - horizontally scalable by adding/removing nodes. Allocate event with diff id to different threads

database - Cassandra support horizontal scaling, like consistent hashing. Add new node - auto rebalance virtual node among all node, no manual shard required

Hotspot issue

some shard/service get more traffic than other
solution - allocate more node to process popular ads, by resource manager

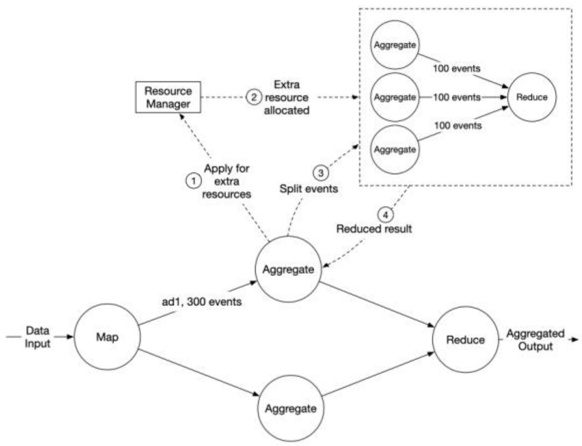

Figure 25 Allocate more aggregation nodes

Fault tolerance

maintain snapshot of system status, and recover from last saved status

<u>Alternate design</u>

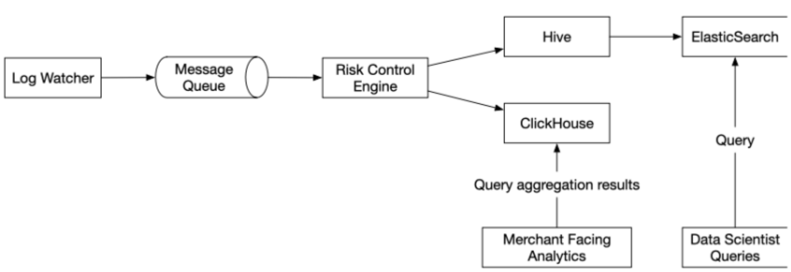

Figure 29 Alternative design

Wrap up

1. Data model and API design
2. Use map reduce paradigm to aggregate ad click events
3. Scale message queue, aggregation service and database
4. Mitigate hotspot issue
5. Monitor system continuously
6. Use reconciliation to ensure correctness
7. Fault tolerance