**High-level design**

**AD-Click aggregator: (streaming infra)**

Spark Job with help of cron job.

Kafka/kinesis “event stream” (events/second) -> use sharding to manage high click numbers. If hot-shard, further partitioning can be done.

* Retention period.

Stream aggregator : flink/spark (fault tolerance with help of retention policy)

* Takes event stream and aggregate them. In memory DS
* Aggregation window: timeframe.
* After timeframe completion, it flushes the value to db(olap)
* Flink : flush interval (interval in which the data is flushed to database) to read partial data. (graphical representation can be viewed).
* Load balancer to take high clicks per second.

Checkpoint on flink 🡪 consider timeframe while checkpoint.

For data-integrity: reconciliation. – dump into a db (S3)

Ad idempotency: ad impression (instance of ad)

Redis (mem-cache): store adImpressionId.

Signed impressionId.

What are Lambda and Kappa architectures?

**Web crawler**

Add urls to frontier queue (KAFKA/SQS)

Avoid giving too much responsibility to crawler.

* Add additional db(dynamo) to store metadata (s3 url link)

Exponential Backoff: have a state on when to run it.

* Add to failed topic(separate) in Kafka (doesn’t support retry) implement by yourself and wait time to retry
* Crawler increases time further based on no. of retries.
* Amazon SQS : supports exponential backoff retries out of the box.
  + Retried once per visibility timeout. Default 30s.
  + Visibility timeout: period other consumers are prevented from receiving message.
    - Increases after each retry failures. Approximate receive count. To move failures to dead letter queue

URL stays in queue unless it is fetched.

Apache Kafka: each message is uniquely identified by offset. CRAWLERS ARE PART OF SAME CONSUMER GROUPS(TO ENSURE EACH URL CONSUMED ONCE).

Amazon SQS : deleted explicitly by Crawler.

Robots.txt:

{

UserAgents:

Disallow: /private/

Crawl-delay:

}

Redis: rate-limiter and DNS Cache.

Efficiency:

Duplicate URL and Duplicate content.

Sharding of URL metadata db based on primary key.

Bloom filter: space efficient, probabilistic DS. HashSet

-false positive.

Crawler-traps: pages on similar domain and falls in the traps.

-maxStep for crawler.

**LeetCode (online judge):**

Monolithic architecture.

Redis (cache)

**Bitly (link shortener)**

Monolithic architecture.

301 redirect: permanent redirect.

302 redirect: temporary redirect.

Unique Short URL:

1. Random generation (1B)

* Base62 ecnoding (0-9,A-Z,a-z).
* Birthday paradox. (2 people with 23 people is 50%)
* We check the database to ensure no collision.

1. Hash (md5) and base62 -> check database.
2. Increment the counter and provide to user

* Predictability (bad security)
* Bijective function (sqids.org) obfuscation.

Low Latency redirect:

* Indexing
* Read-through, lru cache (Redis)
* CDNs the instance to reduce latency. (same issue as 301 vs 302 redirect.)

Scalability:

* 100million = 10^8 users /10^5 (seconds) = 1000 requests/sec. \*(10k) requests => 10^7 requests/sec => 10million/secs.
* Vertical scaling??
* Horizontal scaling -> load balancer. (read scaling as read is major)
* Microservice(read/redirect, write) and API gateway.
* Microservice for small set of code may be overkill.

Database:

500bytes \* 1B -> 500GB

**Live comment (FB/INSTA):**

Functional Requirement:

* Post Comment
* Read comment in realtime (near)
* Read all comments before I joined.

Non-functional Requirement:

* Low latency in comment view.
* Scale (1000 comments/sec)
* Availability of comments.

Core Entities:

* Comments
* Live videos
* Users.

API:

* POST /comments/:videoID
  + {comment: “”}
* GET /comments/:videoID?cursor={last\_id}&page\_size=100
* GET /comments/:videoId?cursor={last\_id}&direction=“before”

High-level design:

* Polling to get the comments(new)
* Getting the comments require persistent connection.

Deep Dives:

* Persistent connection for low latency.
  + Web socket: bidirectional, if new connections come, pushed to user.
    - Doesn’t have http/https protocol.
    - Load balancer, proxy, firewall can break persistent connection.
    - Good for p2p.
  + SSE (SERVER SENT EVENT):
    - Unidirectional
    - Over http.
    - Builtin support.
    - Looks like long http connections.
* Scaling the system:
  + Separate the concerns.
  + Zookeeper as central dispatcher. And cache in write service. AND push the comments.