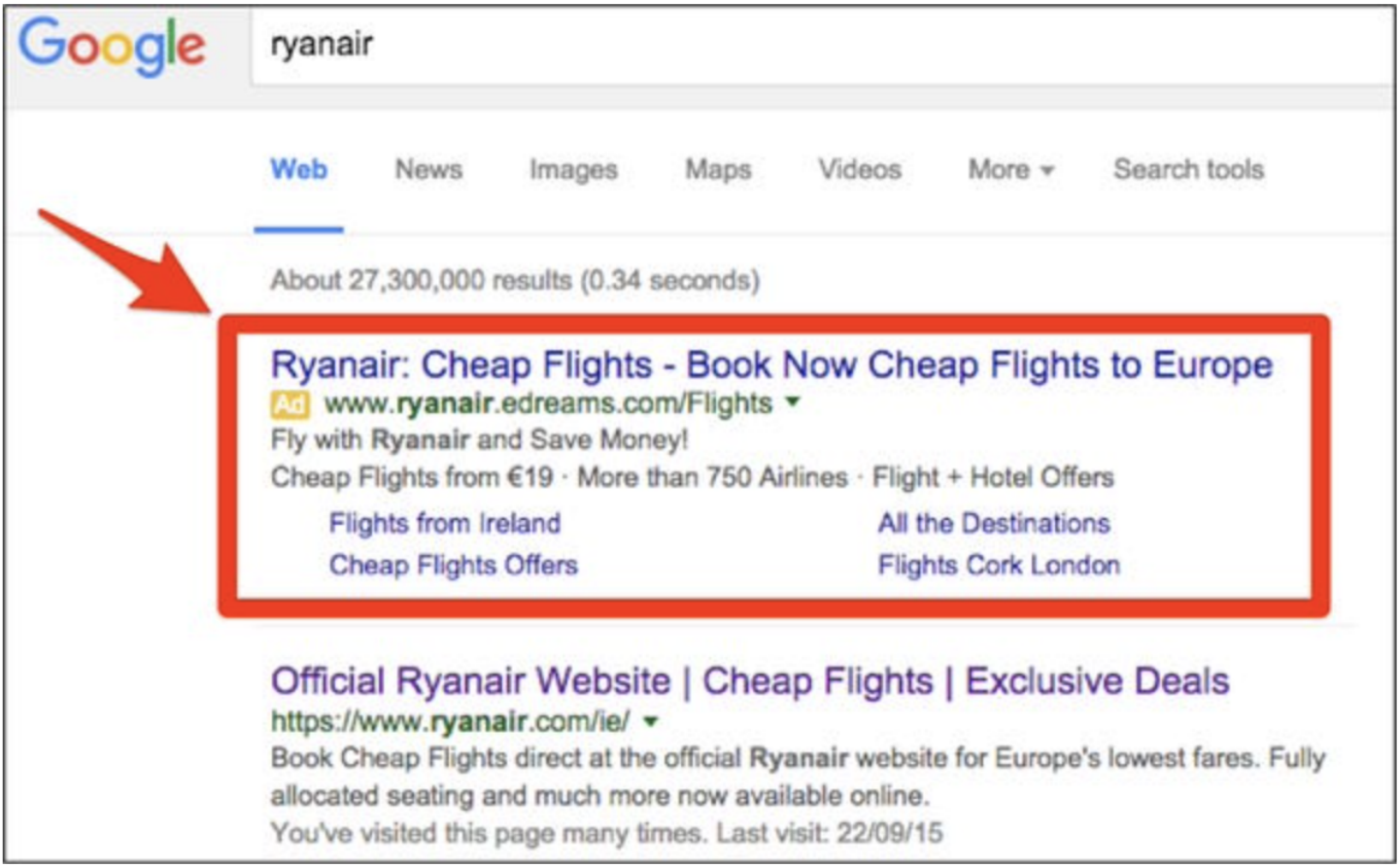
#programming #systemDesign

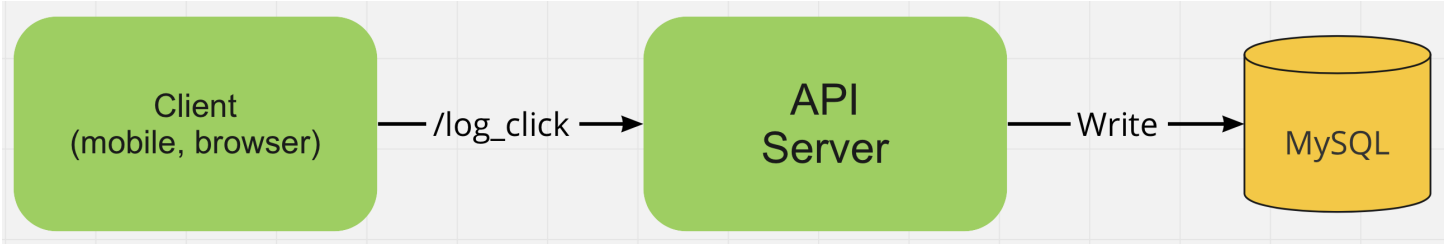
Advertisement Clicks Aggregator



Other Examples

- YouTube video clicks
- Instagram post clicks
- Website clicks
- App clicks

Simple System



- Simple enough design
- Will work for small applications

Will it scale?

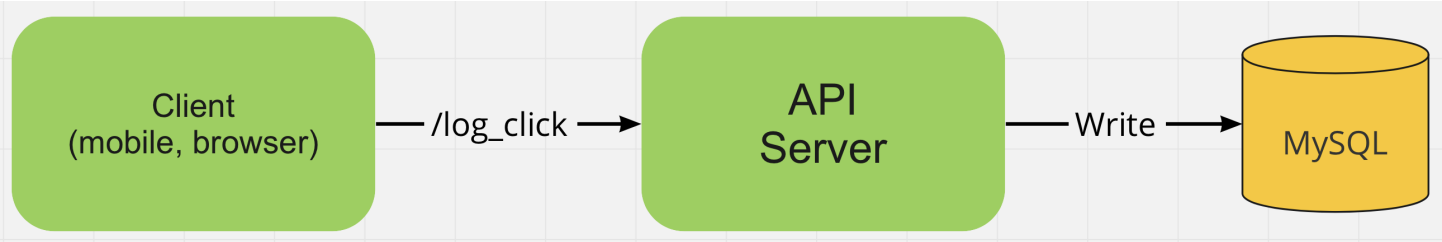
Let's say you have **300M DAU.**
Each user has **10 meaningful clicks** every day

300M * 10 = 3 Billion clicks / day
3B / 24 / 60 / 60 = ~ 35K QPS

Peak QPS = 35K * 2 = 70K QPS

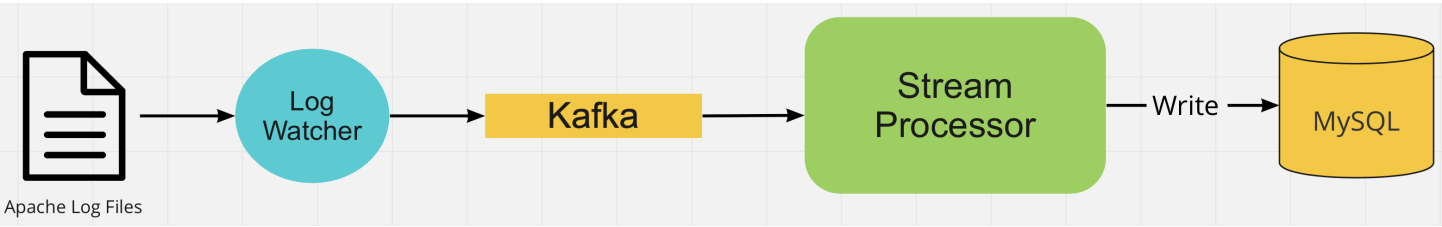
One click data is **0.1KB**
Each day click storage = 0.1KB * 3B = **3TB/day**

Issue 1: Server Overload



- High, Spiky QPS
- Server will need to be overprovisioned
- Synchronous API call more likely to run into errors

So we need to make the system more scalable, elastic & fault tolerant.



- All clicks come to log files
- Watcher to read new logs as they come along
- Writes a message to Kafka with all relevant details

- Lightweight stream processor consumes from Kafka and writes to MySQL

How can we scale Kafka?

- Add more partitions

How can we scale stream processor?

- Add more instances as you add more Kafka partitions
- Bump memory of instance

Issue 2: Storage

Our application requirements:

- High, spiky QPS
- Write intensive
- New data: 3TB/day
- Do we care about storing individual clicks?
  - We care about aggregations more
  - But it's important to keep individual clicks for historical purposes
- Read pattern?
  - Read aggregates such as:
    - Total clicks in last 20 minutes
    - Total clicks in last 30 days
    - Total clicks per advertisements in the last 90 day

Issue 1

High write throughput might introduce too much latency in the system.
Issues with replication lag.
We need a write intensive database.

Issue 2

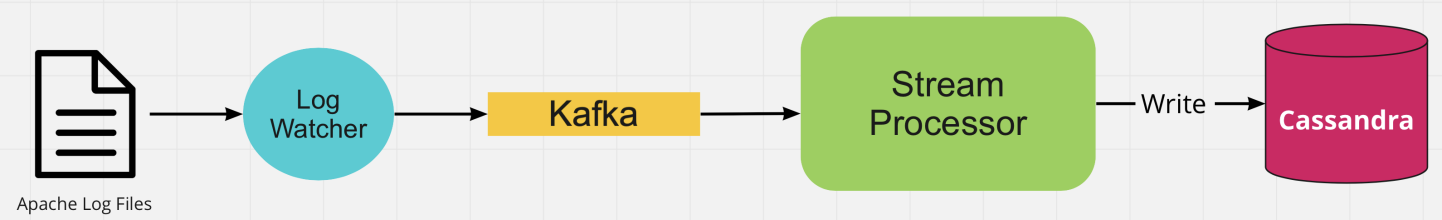
3TB/day means lots of storage consumed per day.
Storing so much data can be expensive.

Issue 3

We mainly care about aggregations, not individual clicks.
Relational DBs won't do well with aggregation queries.

Solution

Use Cassandra.



Cassandra features:

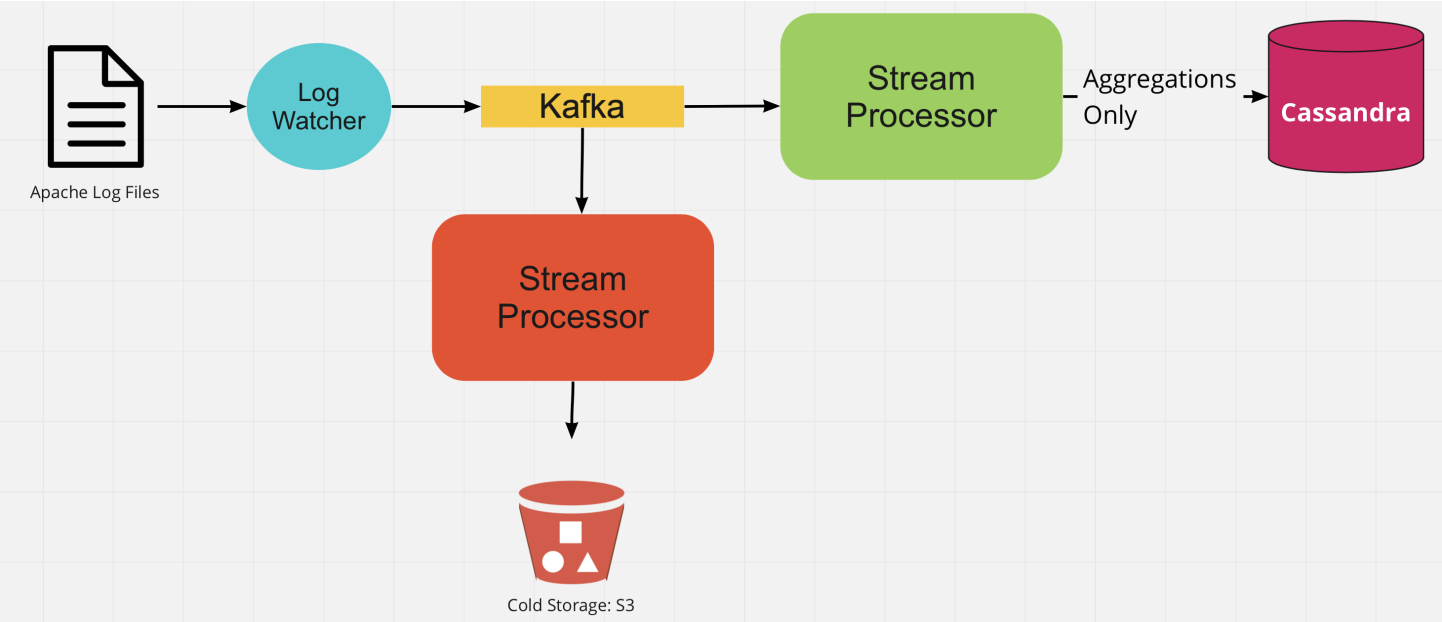
- Handles high write throughput well
- Horizontally scales, so can store huge volume of data easily
- Good for time-series data

Issue 2: Data Model

Where do we store individual clicks?
Where do we store aggregations?

- Aggregation data:
  - Accessed frequently
  - Different time slices
- Individual clicks:
  - Rarely accessed
  - Historical / backfill / backup purposes

Let's not store it in our primary Cassandra table then.



Why use S3 for individual clicks?

- Cheaper to store
- Don't need to access often

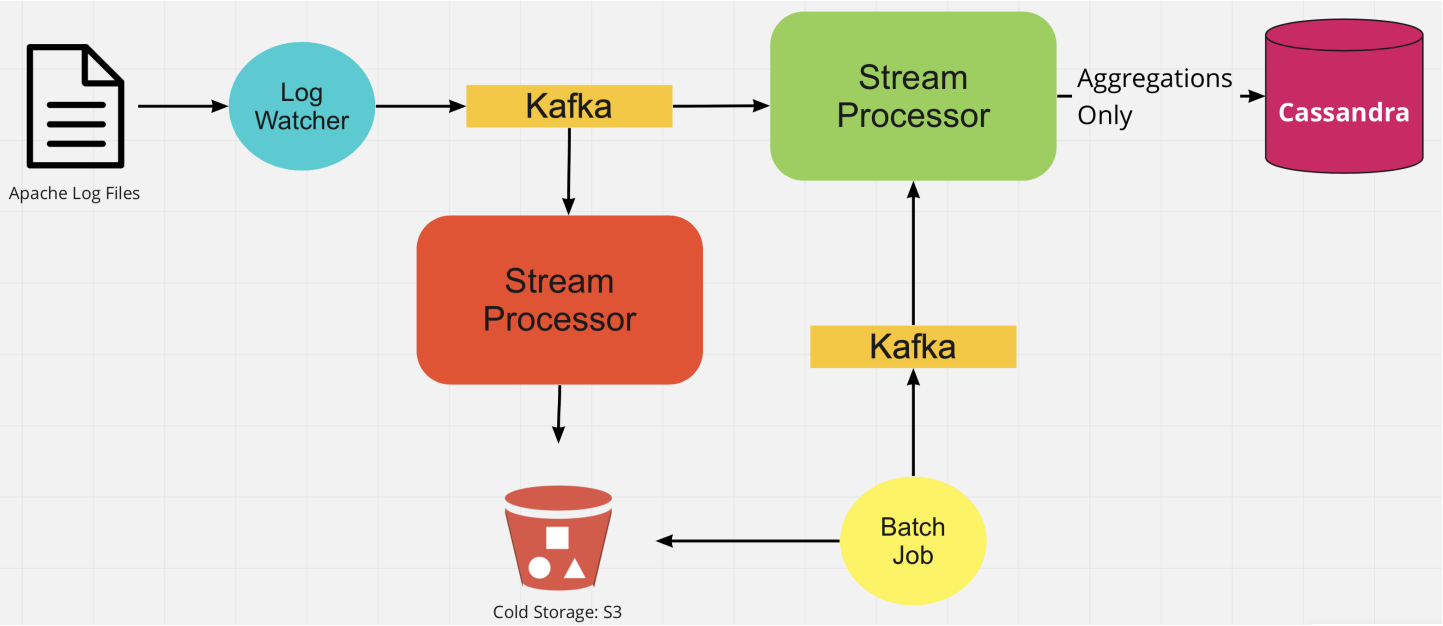
How to store aggregations?

- Different time slices
- Redundant data is okay
- Design Cassandra tables based on query pattern

Issue 3: Data Inconsistency

Stream processors can miss **late data**
Aggregations can be slightly inaccurate

How can we fix that?



Scheduled batch job:

- Reads individual clicks from S3
- Re-publishes them to another Kafka topic
- Goes through stream processor (without late data now)
- Alternatively
    - Batch job directly calculates aggregations and updates Cassandra