My journey in nano "Big-data"

by

Triton Ho 

Content

- Introduction to User Profiling
- Handle massive events
- Suggestion Feed
- User Segmentation

Introduction to User Profiling

Technical Requirement

- HYPEBEAST has > 3M MAU
- Semi-realtime user profiling for Suggestion Feed
- User segmentation

Challenges

- No existing solution
  - Even we can pay
- High traffic, with semi-realtime profiling
- Need integration with existing Wordpress

Simple User Profile

- Just Key-Value pairs
- Higher the value, more interest on the category
- For example:

```
{
    "Nike": 1345.543,
    "Adidas": 5456.873
}
```

Natural Decay of the values

- All love will decay
  - 3 years ago I love BR, now I love CI
- Natural decay is used, due to simplicity
- Problem: What is the optimum half-life?

Final User Profile

- No “optimum” half-life, thus we store values of multiple half-life


```
{
    "Nike": {
        "30": 1345.543,
        "180": 1654.87,
        "360": 3902.654
    },
    "Adidas": {
        "30": 6546.43,
        "180": 8435.432,
        "360": 10254.65
    }
}
```

Physical Storage

- Instead of storing <"userA", "Nike", 30, 1345.543> as a row in database.....
- The user profile can be represented as a json
  - Json is a string~
- For each user, we store ONE row in table user_profile
  - Database schema:
    <userId, jsonString, lastUpdateTime>

Physical Storage, Reasoning

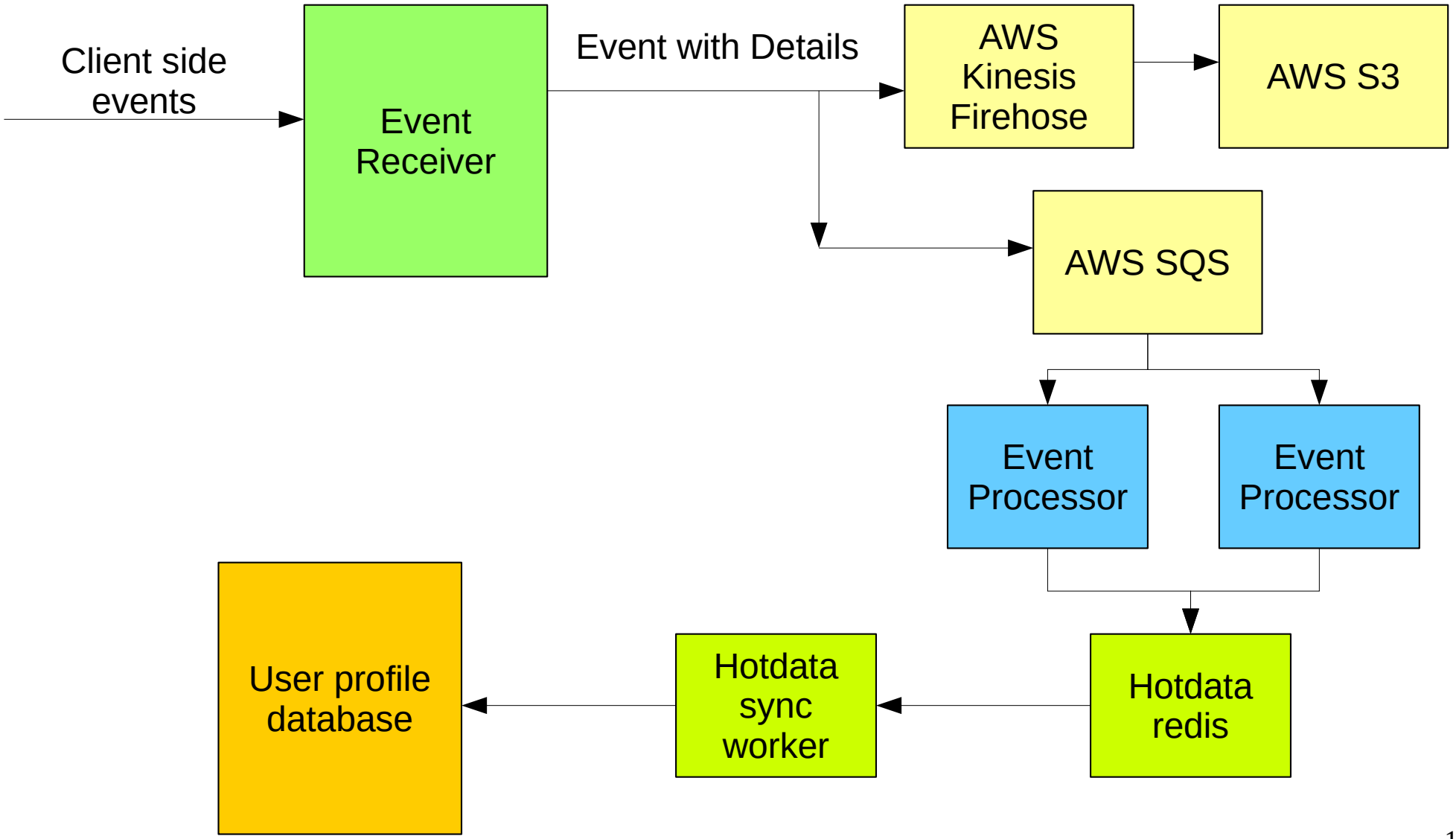
- Suggestion Feed needs whole user profile
  - semi-realtime performance
- User Segmentation needs part of user profile
  - For example:
    Find all user with "Nike"(30 day half-life) > 1000
  - Not realtime
  - Low data freshness requirement
- Store the user profile as one json, means we can get the user profile by ONE logical Random disk IO

Deferred profiles decay

- Instead of update the profiles daily, perform natural decay when the profile is read
- Disk IO at database is hard to scale, but not the CPU at application server

Handle massive events

High level system design



High level system design 1

- Event Receiver
  - 1st tier write buffering
  - unified interface to receive client-side events
  - Input validation and verification
  - Add event details for successive process
  - Pack multiple events to one SQS Message to save money
- Amazon Firehose and S3
  - Store the events for future data mining

High level system design 2

- Amazon SQS
  - 2nd tier of write buffering, providing unlimited queue
  - Distributes the events to Event Processors
- Event Processor
  - Get user profile from database
    - If not exists in redis
  - Update the profile and then put into redis
- Hotdata sync worker
  - Put the dirty user profile from redis back to database
  - For design simplicity, single instance and single thread

Write Aggregation

- Prerequisite
  - The Write is clustered, not random
    i.e. if one record is updated, it is likely to be updated again very soon
  - If some system component crash, some data loss can be tolerated
- Benefits
  - Reduce database write operation
  - Database Write is indepentent of traffic

Write Aggregation in HYPEBEAST

- User will web surfing in our website for a short period
  - i.e. client side events are highly clustered
- If hotdata redis crashed, the data loss is acceptable
- Hotdata sync worker will pick the oldest hotdata in redis first
  - No need to use Sorted Set in redis
  - Pick Top N out of random 100 algorithm is good enough

Optimistic lock

- Two worker may process event of same user
- Need concurrency control to avoid race condition
- Much better performance
  - 1 operation vs 3 operations in pessimistic lock
  - The collision rate is very low

Suggestion Feed

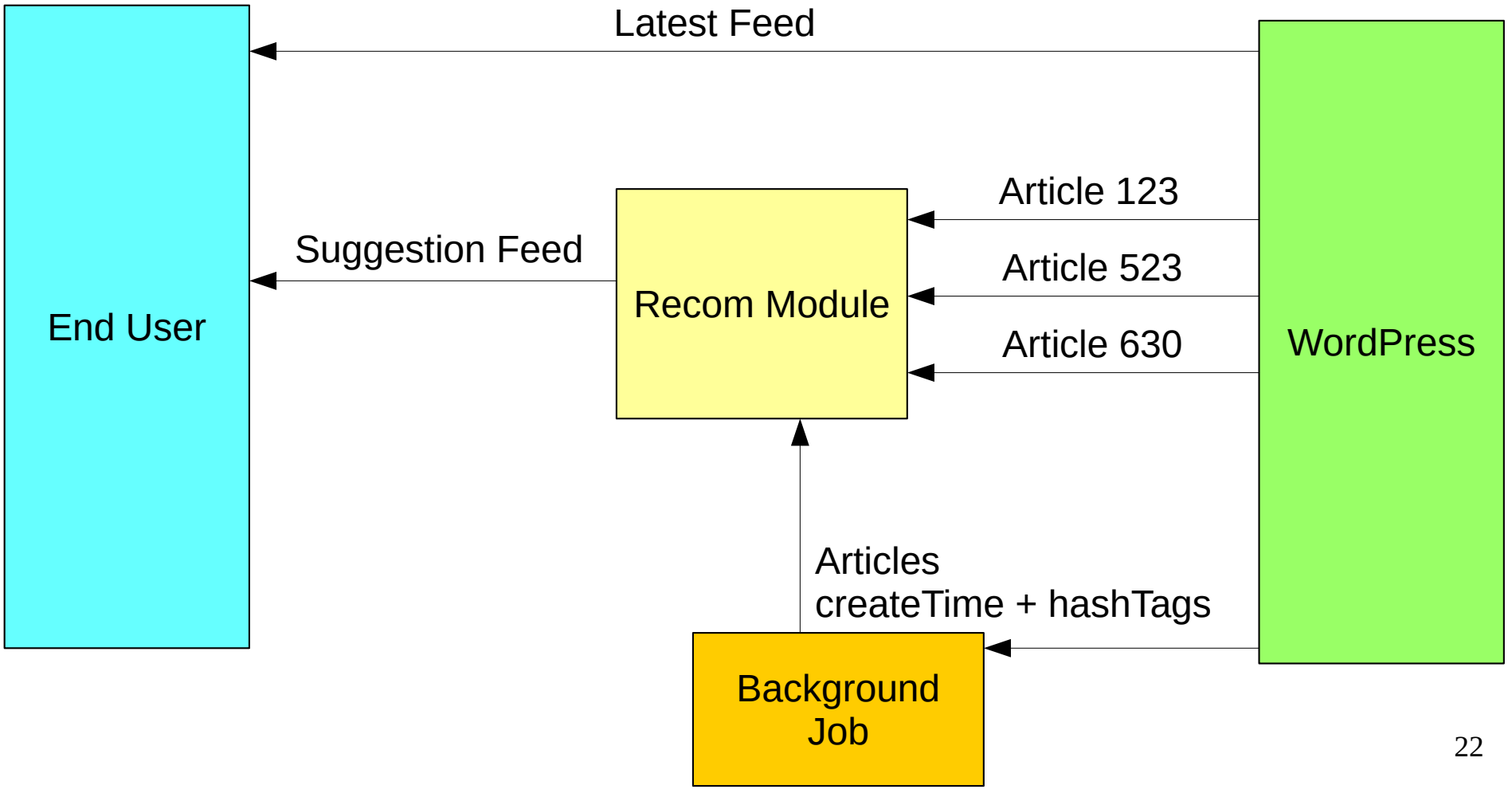
Suggestion Feed

- The articles is stored in Wordpress
- use hashtag to calculate the matchness of the article with user profile
  - We have no linguistic experts
  - We picked ~1000 popular tags
  - Need manual(a.k.a. 工人智慧) tag merging
    - "Nikes" vs "Nike"
    - "iphone 7" vs "iphone"

Integration with Wordpress

- Latest Feed is provided by Wordpress
- Suggestion Feed should be 100% format compatible with Latest Feed
  - Minimize client side development
  - Keep single piece of code
  - future-proof

High level system design



Suggestion Feed workflow

- Recom Module query all articles within X days in local database
- For each candidate article, the hashTags are matched against the user profile to build a score
- For the top N article, Recom Module perform HTTP Request to get back the article content
- Recom Module uses string concatenation of the articles, to simulate the Wordpress Feed format

Suggestion Feed Performance

- All data is cached
  - The user profile, the article content, the article score.....
  - Performance is MUCH more important then data freshness
  - "softcache" library to avoid disaster of cache miss in hot data
- If multiple articles have cache miss, the HTTP Request is performed in multi-thread

User Segmentation

User Segmentation

- A reporting module to answer
  - The histogram of user interest on "Nike"
  - The list of user with "Nike" score > 500

Performance consideration

- In Phase 1, no index avaliable
  - always need full-table scanning
- The json processing for massive user account takes time
  - Multi-thread can help
  - Bottleneck is database network IO and disc IO
- Need sophisticated reporting database, but not enough development time
  - Maybe ElasticSearch?

End~