Recommendation System

INTERVIEW KICKSTART

Functional Requirements

- User activity tracking
  - Track user activity on items
- Online recommendation generation
  - Generate quick and approximate recommendations
- Offline recommendation generation
  - Generate more sophisticated and correct recommendations
- Recommendation dashboard
  - Show recommendation output per user

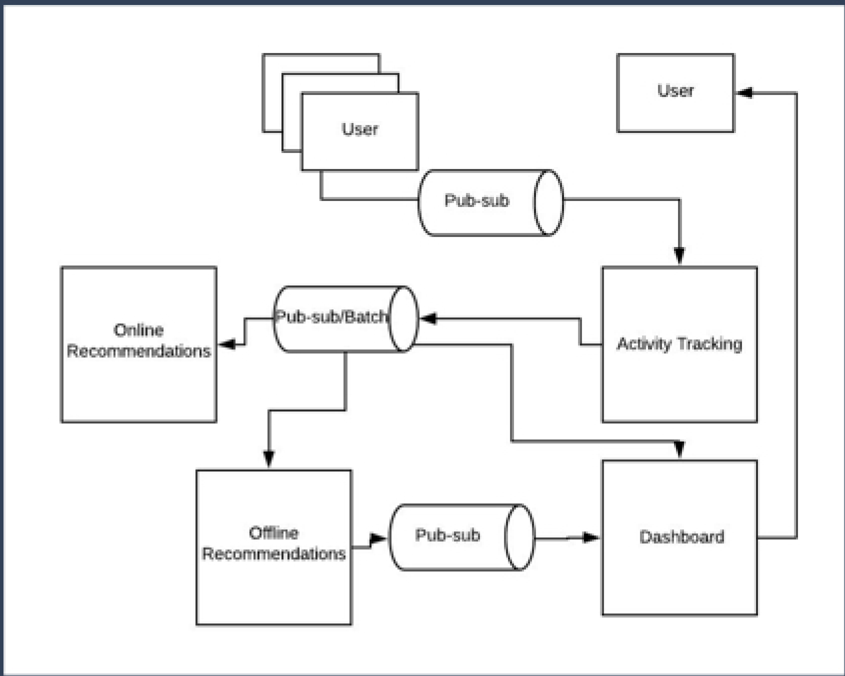
Design Constraints

- Rate of user activities coming in
- Rate of view of recommendation dashboard
- Frequency of execution of offline recommendation

Microservices

- User activity tracking
- Online recommendation generation
- Offline recommendation generation
- Recommendation dashboard

Logical Architecture



User Activity Tracking

INTERVIEW KICKSTART

Logic

- Tiers
  - App server Tier - consumes from pub-sub
  - Cache Tier - for high throughput writes
  - Storage Tier - persistence
- Data Model
  - K: <User id/Item id/auto incrementing key>: V: timestamp, activity type, item genre
- APIs
  - Insert new K-V
- How to store
  - Hashmap in cache
  - Row oriented K-V storage with TTL
    - TTL to determine how long to keep data after generating recommendation
  - Write back caching

INTERVIEW KICKSTART

Need for Scale

- Need to scale for storage: yes
  - Rate of K-V coming in *TTL*size of K-V pair
- Need to scale for throughput: yes
  - Depends on design constraint
- Need to scale for API parallelization: no
  - CRUD APIs do not require parallelization
- Need to scale for removing hotspots: no
  - Insert only data
- Geo-distribution: yes, by user
- Availability: yes

How to Scale

- Architecture: generic (Please refer to any other deck)
- Sharding
  - Horizontal sharding by full key
- Placement of shards in servers
  - Consistent hashing
- Replication
  - Yes
- CP or AP? Does not matter
  - AP is fine

Online Recommendation Generation

Logic

- Similar to trending topics (refer to trending topics deck)
- Tiers:
  - App server tier - to consume events from user activity tracking service
  - Cache or In-memory tier (main tier for all APIs) - for stream computation
  - Storage tier - for recovery purposes
- Data Model:
  - K: V pair
    - K: <user/item genre>: V: ring buffer of 24 counts aggregated per hour
    - K: <genre/item>: V: ring buffer of 24 counts aggregated per hour
  - Store as hashmap in memory
  - Store as row oriented K-V store in storage
  - More space efficient data structure (Count-Min-Sketch) (mainly academic)
  - Priority queue for top N genres per user, top N items per genre
- APIs
  - insertRingBuffer(user, item, genre, timestamp) - updates count
  - computeTrendScore(genre per user) - computes trend score based on sliding window
  - computeTrendScore(item per genre) - computes trend score based on sliding window
  - getTopN()

{ik} INTERVIEW KICKSTART

Need for Scale

- Need to scale for storage ? yes
  - Depends on number of users, items
- Need to scale for throughput ?
  - Ingestion throughput
- Need to scale for API parallelization ?
  - No
- Availability ? yes
- Geo-location based distribution ? no

How to Scale

- Generic architecture
- Sharding
  - Horizontal sharding based on partial key, i.e., user, genre
  - No need for API parallelization as computation is per user or per genre
- Placement of shards in servers
  - Consistent hashing
- Replication
  - Yes for availability as well as for throughput
- CP or AP?
  - AP

Offline Recommendation Generation

Logic

- Tiers:
  - In-memory compute tier
    - copy transformed snapshots in batch from User Activity tracking
    - Run intensive compute algorithms
  - Storage tier
    - Store snapshots in filesystem
  - K: V pair
    - K: <user>: V: list of items user touched
    - K: <item>: V: list of users that touched item (inverse list)
    - K:<item>: V: list of properties
    - Stored in files and retrieved by the compute layer

Algorithms

- Item based collaborative filtering
- Create a full matrix of

Item Id	User 1	User 2	User 3	User 4
Item 1	1	0	1	0
Item 2	1	0	0	1
Item 3	1	1	1	0

(user 1 had clicked on item 1, user 2 never clicked on item2)

- Treat each row as a vector of 1s and 0s and use cosine similarity:
- Compute for every item, the item with closest similarity
- For example, item 1 and item 3 are most similar, so recommend Item 3 for User 2

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

INTERVIEW KICKSTART

Algorithms

- User based collaborative filtering
- Create a full matrix of

User Id	Item 1	Item 2	Item 3	Item 4
User 1	1	0	1	0
User 2	1	0	0	1
User 3	1	1	1	0

- Treat each row as a vector of 1s and 0s and use cosine similarity:
- Compute for every item, the most similar item in terms of property
- User 1 and user 3 are similar, so recommend Item 2 for user 1

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

INTERVIEW KICKSTART

Algorithms

- Content based filtering
- Create a full matrix of

Item Id	Property 1	Property 2	Property 3	Property 4
Item 1	1	0	1	0
Item 2	1	0	0	1
Item 3	1	1	1	0

- Treat each row as a vector of 1s and 0s and use cosine similarity:

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

- Compute for every user, the most similar user
- Item 1 and item 3 are similar in terms of property
- More favorable for cold start for new items, if user x likes item 1, recommend item 3

Need for Scale

- Need to scale for storage: yes
  - Huge amount of transient storage
- Need to scale for throughput: no
  - Daily single jobs
- Need to scale for API parallelization: yes
  - Compute parallelization extremely necessary to scatter gather the generation algorithms
- Need to scale for removing hotspots: no
- Geo-distribution: no
- Availability: no

How to Scale

- Architecture: generic (Please refer to any other deck)
  - Distributed compute layer
  - Distributed file system
- Sharding
  - Distributed file system - horizontally sharded
  - Distributed compute - horizontally sharded
  - Mapreduce on finding cosine similarities parallelly
- Placement of shards in servers
  - Consistent hashing
- Replication
  - Yes
- CP or AP? N/A

INTERVIEW KICKSTART

Recommendation Dashboard

INTERVIEW KICKSTART

Logic

- Tiers
  - App server Tier - consumes from pub-sub
  - Cache Tier - to serve active clients
  - Storage Tier - persistence
- Data Model
  - K: <User id>: V: ring buffer of list of genres and list of movies per genre (simulating a queue)
  - K: <Item Genre>: V: top N Items
- APIs
  - Read(User id), addItem(User id, Item id), addItem(Genre, Top Item)
- How to store
  - Hashmap in cache
  - Row oriented K-V storage
    - maintain most N recent recommendations (fixed per user)
  - LRU caching for active users
- Data is fed from the online recommendation generation as well as offline recommendation generation

Need for Scale

- Need to scale for storage: yes
  - Storage: Number of users *number of items to keep per user*size of item
  - Cache: LRU policy, x% of storage
- Need to scale for throughput: yes
  - Depends on design constraint
- Need to scale for API parallelization: no
  - CRUD APIs do not require parallelization
- Need to scale for removing hotspots: no
- Geo-distribution: yes, by user
- Availability: yes

How to Scale

- Architecture: generic (Please refer to any other deck)
- Sharding
  - Horizontal sharding by full key
- Placement of shards in servers
  - Consistent hashing
- Replication
  - Yes
- CP or AP? Does not matter
  - AP is fine