International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 11 Issue: 05 | May 2024 www.irjet.net p-ISSN: 2395-0072

Building Recommendation Systems Using Lambda Architecture

Vipul Bharat Marlecha[1] , Aqsa Fulara[2], Sreyashi Das[3]

[1]Senior Data Engineer, Netflix USA
[2]Product Manager, Los Angeles, USA
[3]Senior Data Engineer, Netflix USA

Abstract - *Recommendation systems are indispensable tools in today's digital landscape, empowering big tech platforms to deliver personalized experiences to users. This paper delves into the intricate architecture of recommendation systems, exploring their components, data requirements, and the amalgamation of batch processing and streaming data pipelines within the Lambda Architecture framework. We examine various recommendation models, such as collaborative filtering and content-based systems, shedding light on their methodologies and applications. Additionally, we dissect the workflow of recommendation systems, from candidate selection to real-time model serving, elucidating the challenges and strategies encountered at each stage. Through a comprehensive analysis, this paper not only provides insights into the current state of recommendation systems but also anticipates future trends and developments in the field.*

Key Words: recommendation systems, lambda architecture, user preferences, real time analytics

1. Recommendation systems form the backbone of intelligent applications

Recommendations systems have been around for more than a decade. However, with the advent of intelligent systems, personalized recommendations have become increasingly important to build any applications. Think of the last video content, post, or article you saw on your newsfeed, or the product you bought on Amazon, when you exactly didn't know what to consume or buy, how did you land on your final choice? It is likely the application showed you relevant, similar choices that resonated with your nuanced preferences, knowing what you've previously consumed, together with what you're currently exploring. Amazon's 'frequently bought together', or 'deals for you' panels to TikTok's 'For you' feed engaged users to buy their next product or follow the next creator in intuitive and seamless ways. However, building recommendation systems, especially with growing data volumes and ever evolving user preferences is no easy feat.

Recommendation systems are essential tools for enhancing user experience and engagement across various online platforms. Understanding the intricate components and workflow involved in building recommendation systems is vital for designing effective and efficient solutions. This paper provides a comprehensive overview of these components and workflow, elucidating their significance in the recommendation system's architecture.

2. Components of a recommendation system

Whether you are building batch or real-time recommendation systems, there are a few components key to any recommendation systems.

2.1 Input sources for recommendation systems

Input Source 1: Near Real Time Data Source. When users interact with specific items, their actions provide valuable insights to recommendation systems, enabling personalized recommendations. Various interactions such as browsing history, watched content, searched items, purchases, likes, dislikes, clicks, and time spent engaging with content all contribute to understanding how, what, and when users interact with different items. These interactions are captured and stored in user logs, which are ingested into asynchronous processing systems like Apache Kafka. This allows our near-real-time processing pipeline to further clean, process, and feed the data into an online machine learning model.

Input Source 2: Offline Data Source. To achieve highly available, read-heavy data storage for offline data, Cassandra can be a suitable choice due to its distributed architecture and ability to handle large volumes of read requests efficiently. Here's how Cassandra can be utilized for storing the two types of data mentioned.

Items or Content: Cassandra can store the entire inventory of products or content along with their metadata. Each item can be represented as a row in a Cassandra table, with columns representing different attributes such as title, description, categories, genres, themes, images, popularity scores, etc.

The rich metadata associated with each item enables personalized recommendations. Cassandra's ability to handle large amounts of data makes it suitable for storing this comprehensive inventory.

With Cassandra's ability to scale horizontally, it can accommodate the growing inventory of items in ecommerce or content applications efficiently.

User Preferences: Cassandra can also store user preferences and dynamic behavior data. Each user's preferences, browsing behavior, tastes, attributes, etc., can be stored as rows in Cassandra tables.

Real-time updates to user preferences can be handled by Cassandra's ability to support writes at scale. As users interact with the system, their preferences can be updated in real-time, ensuring that the recommendation engine always has the latest data to work with.

Cassandra's distributed nature ensures high availability and fault tolerance, crucial for handling dynamic and unpredictable user behavior effectively.

3. Additional factors influencing recommendation systems

Seasonality Effects. Seasonal trends impact recommendation relevance, requiring consideration in the recommendation algorithm to align suggestions with user interests.

Contextual Data. Geographic, temporal, and social context significantly influence recommendation outcomes, necessitating the incorporation of contextual data for personalized recommendations.

Personal factors. Every person is unique with a variety of interests that change over time and mood-dependent and also the ability to accept or reject recommendations which feeds back into the inference model.

4. Recommendation Models

Collaborative filtering: Seasonal trends impact recommendation relevance, requiring consideration in the recommendation algorithm to align suggestions with user interests.

Context based systems. Here, the models analyze user interactions and item features (genre, style) to recommend items similar to user's past preferences, focusing on individual taste rather than user relationships.

Hybrid Approaches. Many recommendation systems combine collaborative filtering and content-based methods, along with advanced techniques like deep learning, to enhance recommendation accuracy.

Cosine Similarity and Cosine Distance. Cosine similarity is widely used in recommendation systems to find out whether two documents, datasets, images are semantically similar making vector search queries more efficient.

5. Workflow of Recommendation Systems

Candidate Selection: For a given request, either in context of recommendations for specific item and user (like others you may like on a product detail page), or in context of the entire page (home page recommendations), the model starts with sourcing the recommended items. This is generated from all the possible sources of items, considering user interest and item-user affinity.

Ranking Recommendations: The outputs of candidate selection will be a large list of unranked product items that can be possible recommendations. Ranking requires scoring each candidate and sorting them, typically done through large neural networks optimized for positive engagement, revenue maximization, likes or clicks. The outcome is dependent on the model objectives, whether it is to engage users, spend more time on the site or upsell higher priced items.

Filtering and Product features: Once the top 5-10 recommendations are served, most recommendation systems need filtering to serve business logic. Whether it is boosting certain promotional items, or filtering out the out of stock items, the product features are usually completed in post-processing. Heuristics like diverse content or social proof are common to incorporate here.

In ecommerce systems, the recommendation models take the form of 'Frequently bought together' that is typically shown on checkout and add-to-cart pages, while 'Others you may like' and 'Recommended for you' are in the product detail page. The homepage has models like 'Trending', 'Best sellers', 'Top deals' that have a broader appeal.

In content recommendation systems, the homepage models include 'For You', while in-feed models may include 'Similar to this post' or 'Other titles like this'.

6. Real Time Model Serving System

In our recommendation system architecture, Apache Flink plays a pivotal role in processing user interaction data in real-time. This data is sourced from the organization's logging system, which captures input from users across various touchpoints. Upon ingestion into Apache Kafka,

the data stream is then consumed by Apache Flink for further processing and enrichment.

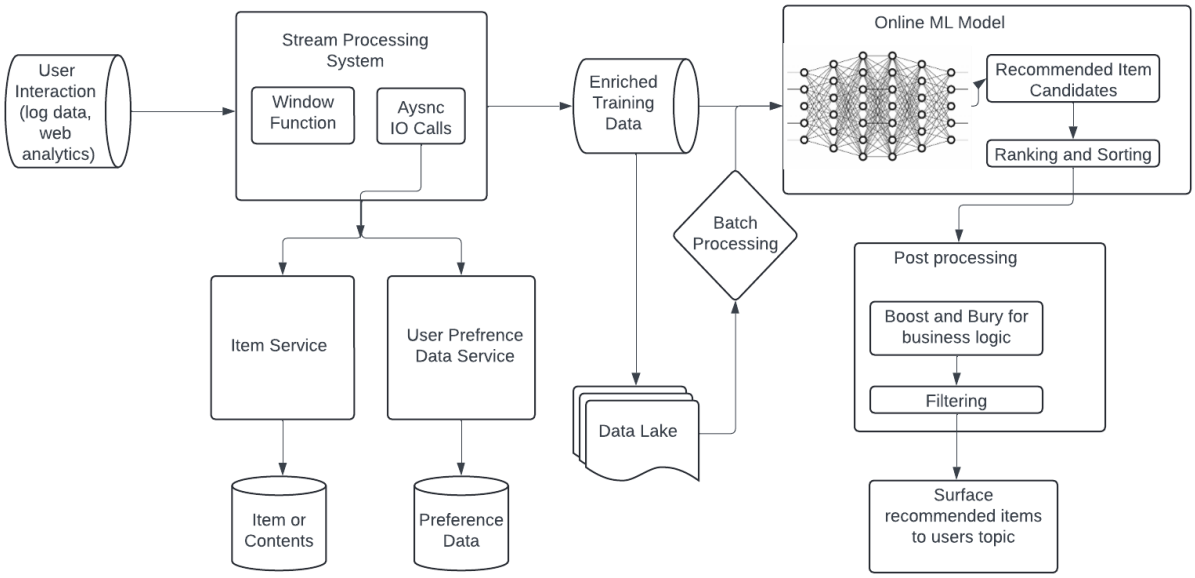
Apache Flink executes three major tasks as part of its real-time processing pipeline.

Windowing based on event time: Apache Flink utilizes event time windowing techniques to organize and process data streams based on the timestamps associated with each event. This approach enables the system to effectively manage and analyze data within specific time intervals, facilitating tasks such as data cleaning, normalization, and deduplication. By partitioning data into windows, Apache Flink ensures that data processing operations are performed efficiently and accurately, leading to improved data quality and reliability.

Asynchronous IO calls to Item Service and User preferences service: Apache Flink interacts asynchronously with external services, such as the Item Service and User Preferences Service, to retrieve additional metadata and dynamic behavior data related to items and user preferences. These IO calls enable the system to enrich the raw data stream with valuable contextual information, enhancing the quality and relevance of recommendations generated by the system. By leveraging external services in an asynchronous manner, Apache Flink ensures that data processing operations remain efficient and non-blocking, thereby minimizing latency and maximizing throughput.

Sink Data to Kafka: Once data processing and enrichment tasks are completed, Apache Flink sinks the processed data back into Apache Kafka for further consumption or storage. This final step ensures that the enriched data is seamlessly integrated back into the streaming pipeline, making it available for downstream applications or systems. By leveraging Kafka as a data sink, Apache Flink enables seamless integration with other components of the recommendation system architecture, facilitating data exchange and interoperability across different modules or services.

7. Architecture



8. Key Challenges in Real Time Recommendations and how to solve them

8.1. Cold Start Problem: When a new user accesses the application, the recommendation system lacks sufficient data about their preferences or interactions. This issue, known as the cold start problem, hampers the system's ability to provide personalized recommendations. To mitigate this challenge, the system can resort to various strategies:

Fallback on Item-Based Recommendations. In the absence of user-specific data, the system can rely on item-based information to make recommendations. This involves suggesting items that are popular or similar to those frequently accessed by other users.

Utilization of Generic Information. Leveraging generic data from browser-related metadata, such as location or device type, can help personalize recommendations to some extent. By inferring user preferences based on demographic or contextual information, the system can provide initial recommendations tailored to the user's likely interests.

8.2. Sparse Interactions: Limited user interactions pose a challenge in gathering sufficient data to make accurate recommendations. Sparse interactions hinder the system's ability to understand user preferences and effectively personalize recommendations. To address this challenge, the system can adopt several strategies:

Content-Based Recommendations. By analyzing the attributes of items and comparing them to user preferences, the system can make recommendations based on content similarity. This approach reduces reliance on user interactions and enhances recommendation accuracy, particularly in scenarios with sparse user data.

Implicit Feedback Modeling. Instead of relying solely on explicit user interactions (e.g., ratings or reviews), the system can incorporate implicit feedback signals such as clicks or dwell time. By inferring user preferences from implicit signals, the system can mitigate the impact of sparse interactions and improve recommendation quality.

8.3. Large Volumes of Streaming Data / Noisy Data for Real-Time Interactions: Handling large volumes of streaming data and dealing with noise present significant challenges in real-time recommendation systems. The influx of data can overwhelm the system and introduce inaccuracies in recommendations. To address this challenge, the system can implement the following strategies:

Streamlining Data Processing Pipelines. Optimizing data processing pipelines can improve the system's ability to handle large volumes of streaming data efficiently. Techniques such as parallel processing, distributed computing, and stream buffering can enhance scalability and performance.

Noise Reduction Techniques. Implementing noise reduction techniques such as data filtering, outlier detection, and anomaly detection can help improve data quality and minimize the impact of noisy data on recommendation accuracy. By identifying and mitigating noisy data, the system can enhance the reliability of real-time recommendations.

8.4. Data Skew: Data skew, characterized by uneven distribution of data across partitions or processing nodes, can introduce performance bottlenecks and affect the scalability of real-time recommendation systems. To mitigate data skew challenges, the system can adopt the following approaches:

Partitioning and Load Balancing. Implementing effective data partitioning strategies and load balancing techniques can distribute data evenly across processing nodes or partitions. By evenly distributing the workload, the system can alleviate data skew issues and improve system performance.

Dynamic Scaling. Employing dynamic scaling mechanisms that automatically adjust resource allocation based on workload fluctuations can help address data skew challenges. By dynamically scaling resources in response to changing data distribution patterns, the system can maintain optimal performance and scalability.

9. Data Quality in Recommendation Systems

The quality of data has a major impact in avoiding “trust busters” and improving recommendations. During the design of the feature store three important factors can be taken into account while evaluating which features or dimensions to include 1) timeliness 2) coverage 3) accuracy. Essentially data completeness per item, per user and per feature can be considered before productionalizing the feature store. A highly diverse feature set may not positively influence the recommendations. In addition to the quantitative measures, it is also critical to gather qualitative feedback from users about their satisfaction with the recommendations. This can be done through user surveys, interviews, or user testing sessions.

10. Conclusion

Sophisticated recommendation systems will be built incorporating more context like location, time and even knowing the users current mood. There are many more ideas that are yet to be tried like conversational recommendations while searching for movies to watch or simply having a robot cook your favorite meal. There is growing awareness of the significance of fairness and bias in recommendation systems and systems that detect bad data can help in generating more equitable recommendations.

REFERENCES

[1] Kartik Chandra Jena, Sushruta Mishra, Soumya Sahoo and Brojo Kishore Mishra. 2017. "Principles, techniques and evaluation of recommendation systems". 2017 International Conference on Inventive Systems and Control (ICISC). doi:10.1109/ICISC.2017.8068649

[2] T. H. Roh, K. J. Oh, and I. Han. The collaborative filtering recommendation based on some cluster-indexing cbr. Expert systems with applications, 25(3):413–423, 2003.

[3] S.M. Mahdi Seyednezhad, Kailey Nobuko Cozart, John Anthony Bowllan, Anthony O. Smith. 2018. "A Review on Recommendation Systems: Context-aware to Social-based". arXiv:1811.11866

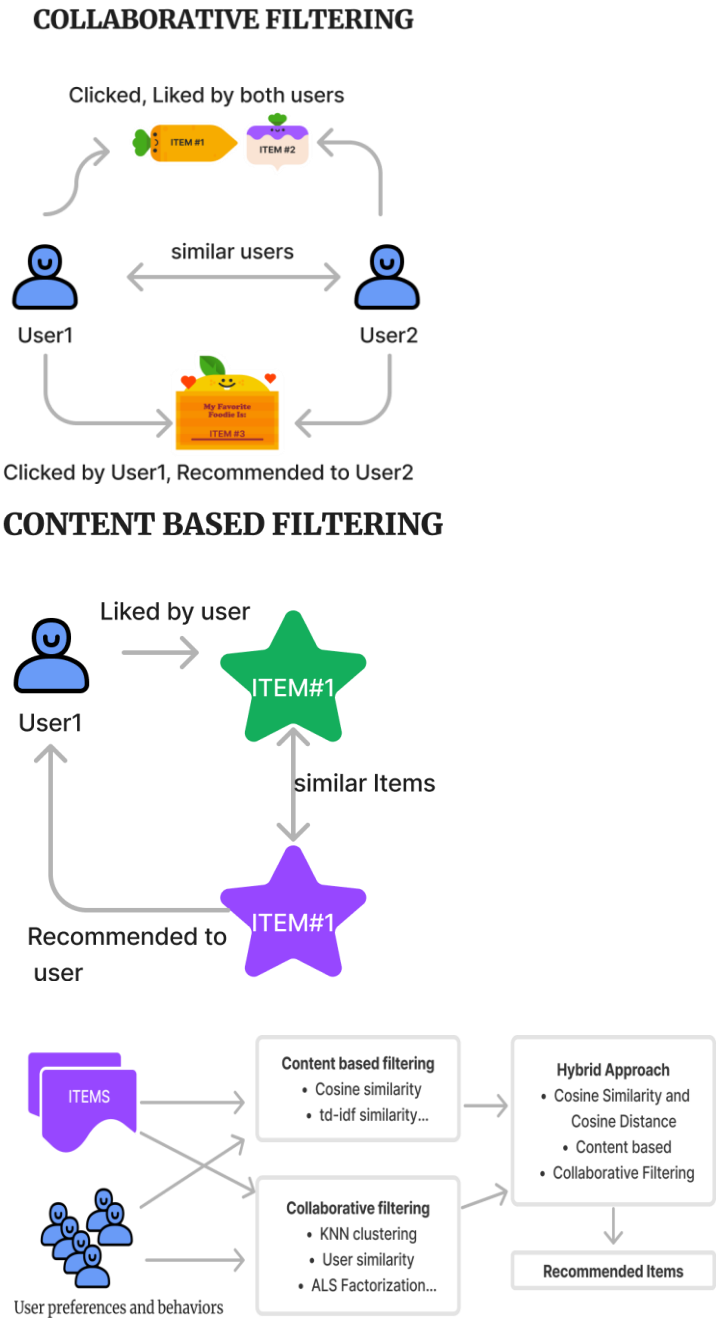
[4] Matteo Marcuzzo, Alessandro Zangari, Andrea Albarelli and Andrea Gasparetto. 2022. "Recommendation Systems: An Insight Into Current Development and Future Research Challenges", IEEE Access,Volume 10, Pages 86578 - 86623, doi: 10.1109/ACCESS.2022.3194536

[5] N. Nikzad–Khasmakhi, M.A. Balafar and M. Reza Feizi–Derakhshi. 2019. "The state-of-the-art in expert recommendation systems". Engineering Applications of Artificial Intelligence Volume 82, June 2019, Pages 126-147

[6] R. Burke. "Evaluating the dynamic properties of recommendation algorithms". In ACM RecSys, 2010

[7] Gediminas Adomavicius and Jingjing Zhang.2012. "Impact of data characteristics on recommender systems performance". ACM Transactions on Management Information SystemsVolume 3Issue 1Article No.: 3pp 1–17

[8] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. 2017. "Streaming recommender systems". In Proceedings of the 26th international conference on world wide web. 381--389



Factor	Collaborative Filtering	Content-Based Recommendations
Approach	Utilizes user-item interactions and user similarities to generate recommendations.	Focuses on item features and user preferences to recommend similar items.
Data Requirements	Requires a large amount of user interaction data (e.g., ratings, views).	Needs detailed information on item attributes and user preferences.
Advantages	Can recommend items without understanding their content. Effective at discovering new interests.	Does not require data from other users. Can be more transparent in how recommendations are made.
Disadvantages	Suffers from cold start problem for new users and items Can be less interpretable as it does not consider item content.	Limited to recommending items similar to previous likes (low personalization).