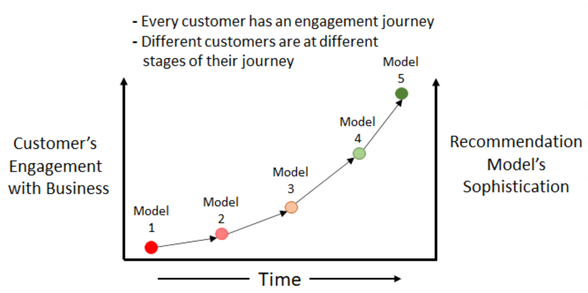
Scalable High-Performance Architecture for Evolving Recommender System

Ravi Kumar Singh [1] Mayank Mishra [1] Rekha Singhal [1]

[1]TCS Research

Introduction and Motivation



Over time, customers develop different engagement points with the business. As shown in Figure 1, an organization may use multiple models (of varying levels of sophistication) in their recommendation system with increased customer engagement. This means the recommendation systems[2] of businesses evolve with the introduction of newer, more sophisticated models and the removal of old models. Moreover, multiple recommender models coexist at the same time.

Following are the technical debts associated with deploying multiple recommender models simultaneously.

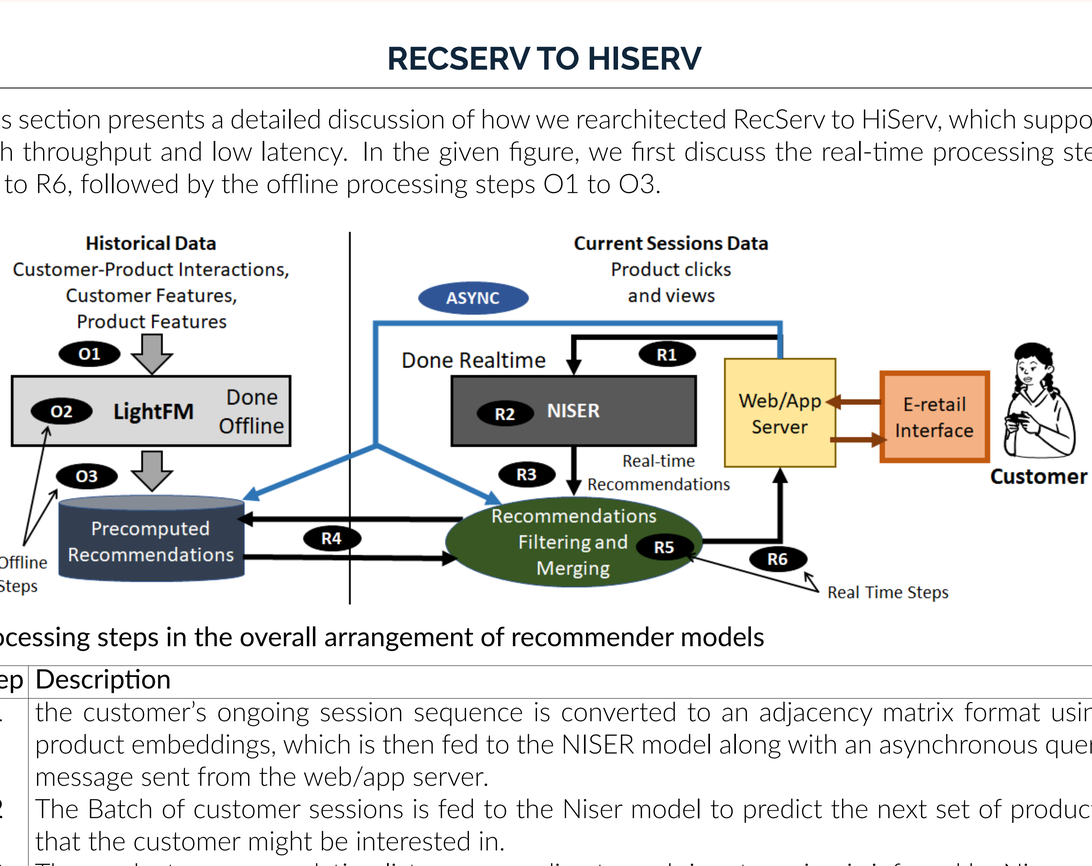
- Requirements: It can be offline or real-time for models.
- Technology: Models can be ML-based, or others can be DL based.
- Architecture: It can differ in on-prem or cloud-based models.
- Data Features: Data features used during training and inference can differ.

Problem Definition: we re-architect an in-house recommender the system named RecServ, to create HiServ (High-performance recommendation Serving), which employs multiple recommender models and recommends items with low latency, offers high throughput and handles huge volumes of data over time.

RECOMMENDER SYSTEM USE CASE

HiServ is a personalized1 recommendation system designed for a clothing retail website that caters to almost 2 million known customers and covers over 4000 product categories. Categories have an average of 100 products, ranging from 50 to 1000. The system uses customer browsing, purchase history, and recently viewed products to personalize product listings.

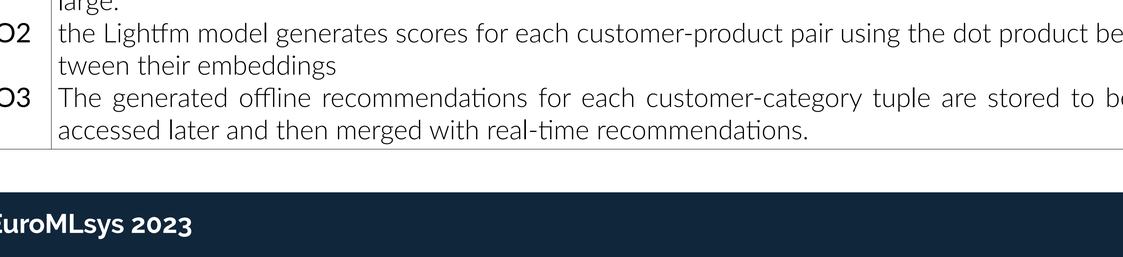
- **Performance Requirements:** the system demands an end-to-end latency requirement of 60ms, with the end being the point where the request for recommendation is generated
- **Recommendation Architecture:** The recommender system under discussion involved two recommendation models, namely



- **LightFM model** is an ML the based model uses customer features (age, gender, etc.), product features (price, categories, etc.), and customer-product interaction features (buys, views, reviews, etc.) for learning Embeddings for customers and products. This model performs dot product over the learned Customer Embeddings and Product Embeddings and calculates the scores for every product corresponding to each category and customer.
- **NISER model**[1] provides real-time session-based recommendations of products using a customer's past clicks in the current session(sequence of product clicks/views within a given time duration).

RECSERV TO HISERV

This section presents a detailed discussion of how we rearchitected RecServ to HiServ, which supports high throughput and low latency. In the given figure, we first discuss the real-time processing steps R1 to R6, followed by the offline processing steps O1 to O3.



Processing steps in the overall arrangement of recommender models

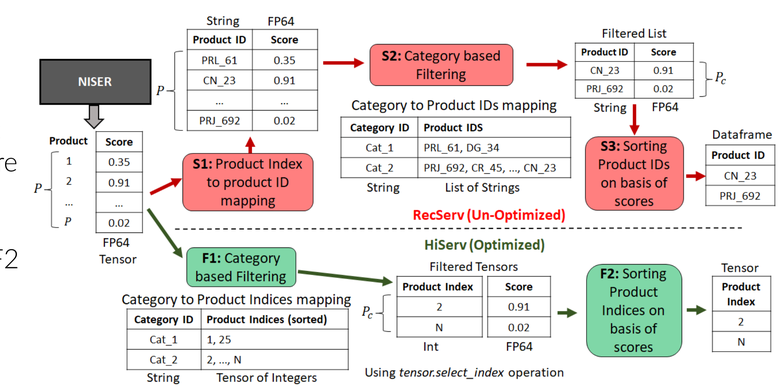
Step	Description
R1	the customer's ongoing session sequence is converted to an adjacency matrix format using product embeddings, which is then fed to the NISER model along with an asynchronous query message sent from the web/app server.
R2	The Batch of customer sessions is fed to the Niser model to predict the next set of products that the customer might be interested in.
R3	The product recommendation list corresponding to each input session is inferred by Niser
R4	the product list recommended by Lightfm is accessed from the cache when the asynchronous query message is received from the web/app server.
R5	the recommendation lists of both Niser and Lightfm are merged to create the final recommendation list of products
R6	Final recommendation list of products provided to the web/App server.
O1	Batch Preparation for LightFM model inference. The idea is to create batches of customers and categories of products as $2D$ vectors of both customers and product embeddings become very large.
O2	the Lightfm model generates scores for each customer-product pair using the dot product between their embeddings
O3	The generated offline recommendations for each customer-category tuple are stored to be accessed later and then merged with real-time recommendations.

Challenges in Processing steps in HiServ

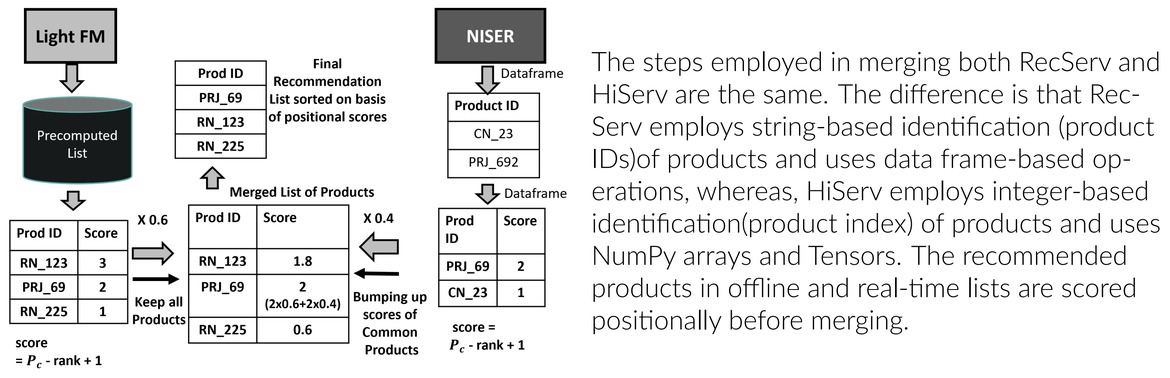
Step	Challenges
R1	How many sessions to batch together and creation of batch waiting of enough requests to accumulate?
R2	How fast the NISER model can provide inference?
R3	How best can we prepare the lists for merging with the recommendations list predicted by Lightfm?
R4	How to arrange data in the cache and which cache technology to use?
R5	What kind of data format and Data structures do us to make the process of merging fast?
R6	How to cache them for future use?
O1	Create batches of customers and categories of products as $2D$ vectors of both customers and product embeddings such that it should not become very large.
O2	Goal is to perform the dot product of such large embeddings quickly.
O3	How to put generated data in the cache?

R3: Processing of NISER Recommendations Un-optimized vs Optimized

The steps S1, S2, and S3 performed by RecServ using product IDs (string-based identification) are replaced by faster Tensor-based operations using product index (integer-based identification)F1, F2 steps.



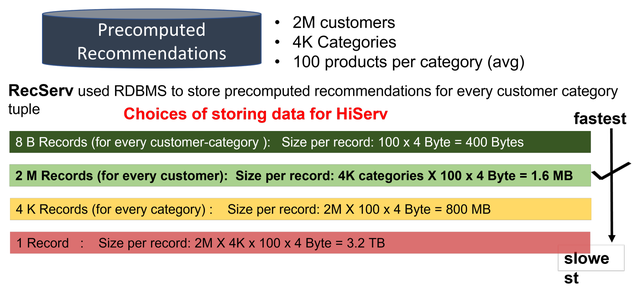
R5- Offline and real-time recommendation merging speedup



The steps employed in merging both RecServ and HiServ are the same. The difference is that RecServ employs string-based identification (product IDs)of products and uses data frame-based operations, whereas, HiServ employs integer-based identification(product index) of products and uses NumPy arrays and Tensors. The recommended products in offline and real-time lists are scored positionally before merging.

O3- Storing offline recommendations

RecServ relied on RDBMS solely for read-only purposes. Accessing the precomputed recommendations using an API for each customer-category tuple resulted in high latency, taking hundreds of milliseconds (300 to 800 milliseconds). However, HiServ's light green box with a tick mark resolves this issue by storing the product recommendation list as a single record in a key-value store for each customer.



EVALUATION

Using better techniques ensured that the performance of HiServ was much better than RecServ. Table 1 shows that latency per recommendation supported by HiServ is close to 65ms, which is roughly 23× lesser than RecServ. The throughput of HiServ is close to 1500 requests per second.Batching plays an essential role in the high performance achieved by HiServ.

Experimental Setup: The underlying hardware of a 16-core 64 GB VM where we performed experiments has 16-core 64 GB VM.

Table 1. Latency and Throughput For HiServ vs RecServ

Recommender System	Latency	Throughput
RecServ	1500 ms	1 per sec
HiServ (batch size 100)	65 ms	1538 per sec
HiServ (batch size 50)	40 ms	1250 per sec
HiServ (batch size 10)	16 ms	625 per sec
HiServ (batch size 5)	11 ms	454 per sec
HiServ (batch size 1)	5.5 ms	181 per sec

Conclusions

People are using different recommendation models for recommendations that are evolving. Recommendation models start degrading in performance due to technical debt. In this paper, we have looked at our in-house case study of two mod- els, which are LightFM(ML model) and Niser. (DL model). Our in-house recommender system has been re-architected into HiServ, which utilizes various data optimization techniques such as data purging, operation fusion, and concurrent user management, as well as hashmap, NumPy, and tensors operations. These methods result in low recommendation latency and high throughput. HiServ recommends the next likely products for multiple customers by bundling them in batches, reducing latency from 1.5 seconds to 65 milliseconds and enhancing throughput from 1 recommendation per second to 1500 recommendations per second.

References

[1] Priyanka Gupta, Diksha Garg, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. Niser: Normalized item and session representations with graph neural networks. 09 2019.

[2] Richa Sharma and Rahul Singh. Evolution of recommender systems from ancient times to modern era: A survey. *Indian Journal of Science and Technology*, 9, 05 2016.